



Middle East Technical University Northern Cyprus Campus
Computer Engineering Program

CNG491 Computer Engineering Design I

Markopy Documentation

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

Supervised by
Assoc. Prof. Dr. Okan Topçu

0.4.1 Documentation

1 Markov Passwords	3
1.1 About The Project	3
1.1.1 Built With	3
1.2 Getting Started	3
1.2.1 Prerequisites	4
1.2.2 Installing Dependencies	4
1.2.3 Installation	4
1.2.4 Building	4
1.3 Linux	4
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	5
1.5.3 Contributing	6
1.5.4 Contact	6
2 Deprecated List	7
3 Namespace Index	9
3.1 Namespace List	9
4 Hierarchical Index	11
4.1 Class Hierarchy	11
5 Class Index	13
5.1 Class List	13
6 File Index	15
6.1 File List	15
7 Namespace Documentation	17
7.1 markopy_cli Namespace Reference	17
7.1.1 Detailed Description	17
7.1.2 Function Documentation	17
7.1.2.1 cli_generate()	17
7.1.2.2 cli_init()	18
7.1.2.3 cli_train()	18
7.1.3 Variable Documentation	19

7.1.3.1 action	19
7.1.3.2 args	19
7.1.3.3 bulk	19
7.1.3.4 corpus_list	19
7.1.3.5 default	19
7.1.3.6 help	19
7.1.3.7 model	19
7.1.3.8 model_base	19
7.1.3.9 model_extension	20
7.1.3.10 model_list	20
7.1.3.11 output	20
7.1.3.12 output_file_name	20
7.1.3.13 output_forced	20
7.1.3.14 parser	20
7.1.3.15 True	20
7.2 Markov Namespace Reference	20
7.2.1 Detailed Description	21
7.3 Markov::API Namespace Reference	21
7.3.1 Detailed Description	21
7.4 Markov::API::CLI Namespace Reference	21
7.4.1 Detailed Description	22
7.4.2 Typedef Documentation	22
7.4.2.1 ProgramOptions	22
7.4.3 Function Documentation	22
7.4.3.1 operator<<()	22
7.5 Markov::API::Concurrency Namespace Reference	22
7.5.1 Detailed Description	22
7.6 Markov::API::CUDA Namespace Reference	22
7.6.1 Detailed Description	23
7.7 Markov::GUI Namespace Reference	23
7.7.1 Detailed Description	23
7.8 Markov::Markopy Namespace Reference	23
7.8.1 Function Documentation	23
7.8.1.1 BOOST_PYTHON_MODULE()	23
7.9 Markov::Random Namespace Reference	24
7.9.1 Detailed Description	24
7.10 model_2gram Namespace Reference	24
7.10.1 Detailed Description	24
7.10.2 Variable Documentation	24
7.10.2.1 alphabet	25
7.10.2.2 f	25
7.11 random Namespace Reference	25

7.11.1 Detailed Description	25
7.12 random-model Namespace Reference	25
7.12.1 Variable Documentation	25
7.12.1.1 alphabet	25
7.12.1.2 f	25
7.13 Testing Namespace Reference	25
7.13.1 Detailed Description	26
7.14 Testing::MarkovModel Namespace Reference	26
7.14.1 Detailed Description	26
7.14.2 Function Documentation	26
7.14.2.1 TEST_CLASS() [1/3]	26
7.14.2.2 TEST_CLASS() [2/3]	27
7.14.2.3 TEST_CLASS() [3/3]	28
7.15 Testing::MarkovPasswords Namespace Reference	29
7.15.1 Detailed Description	29
7.16 Testing::MVP Namespace Reference	29
7.16.1 Detailed Description	29
7.17 Testing::MVP::MarkovModel Namespace Reference	29
7.17.1 Detailed Description	30
7.17.2 Function Documentation	30
7.17.2.1 TEST_CLASS() [1/3]	30
7.17.2.2 TEST_CLASS() [2/3]	31
7.17.2.3 TEST_CLASS() [3/3]	32
7.18 Testing::MVP::MarkovPasswords Namespace Reference	35
7.18.1 Detailed Description	35
7.18.2 Function Documentation	35
7.18.2.1 TEST_CLASS()	35
8 Class Documentation	39
8.1 Markov::API::CLI::_programOptions Struct Reference	39
8.1.1 Detailed Description	41
8.1.2 Member Data Documentation	41
8.1.2.1 bExport	41
8.1.2.2 bFailure	41
8.1.2.3 blmport	41
8.1.2.4 datasetname	41
8.1.2.5 exportname	41
8.1.2.6 generateN	41
8.1.2.7 importname	41
8.1.2.8 outputfilename	42
8.1.2.9 seperator	42
8.1.2.10 wordlistname	42

8.2 Markov::GUI::about Class Reference	42
8.2.1 Detailed Description	43
8.2.2 Constructor & Destructor Documentation	43
8.2.2.1 about()	43
8.2.3 Member Data Documentation	43
8.2.3.1 ui	43
8.3 Markov::API::CLI::Argparse Class Reference	44
8.3.1 Detailed Description	45
8.3.2 Constructor & Destructor Documentation	45
8.3.2.1 Argparse() [1/2]	45
8.3.2.2 Argparse() [2/2]	45
8.3.3 Member Function Documentation	47
8.3.3.1 getProgramOptions()	47
8.3.3.2 help()	47
8.3.3.3 parse()	48
8.3.3.4 setProgramOptions()	48
8.3.4 Member Data Documentation	49
8.3.4.1 po	49
8.4 Markov::GUI::CLI Class Reference	49
8.4.1 Detailed Description	50
8.4.2 Constructor & Destructor Documentation	50
8.4.2.1 CLI()	50
8.4.3 Member Function Documentation	50
8.4.3.1 about	51
8.4.3.2 start	51
8.4.3.3 statistics	51
8.4.4 Member Data Documentation	51
8.4.4.1 ui	51
8.5 Markov::API::CUDA::CUDADeviceController Class Reference	51
8.5.1 Detailed Description	53
8.5.2 Member Function Documentation	53
8.5.2.1 CudaCheckNotifyErr()	53
8.5.2.2 ListCudaDevices()	53
8.6 Markov::API::CUDA::CUDAModelMatrix Class Reference	53
8.6.1 Detailed Description	57
8.6.2 Member Function Documentation	57
8.6.2.1 AdjustEdge()	57
8.6.2.2 ConstructMatrix()	58
8.6.2.3 CudaCheckNotifyErr()	59
8.6.2.4 DumpJSON()	60
8.6.2.5 Edges()	60
8.6.2.6 Export() [1/2]	60

8.6.2.7 Export() [2/2]	61
8.6.2.8 FastRandomWalk()	61
8.6.2.9 FastRandomWalkPartition()	62
8.6.2.10 FastRandomWalkThread()	63
8.6.2.11 Generate()	64
8.6.2.12 GenerateThread()	65
8.6.2.13 Import() [1/2]	66
8.6.2.14 Import() [2/2]	67
8.6.2.15 ListCudaDevices()	68
8.6.2.16 MigrateMatrix()	68
8.6.2.17 Nodes()	68
8.6.2.18 OpenDatasetFile()	68
8.6.2.19 RandomWalk()	69
8.6.2.20 RetrieveCudaBuffer()	70
8.6.2.21 Save()	70
8.6.2.22 StarterNode()	71
8.6.2.23 Train()	71
8.6.2.24 TrainThread()	72
8.6.3 Member Data Documentation	73
8.6.3.1 datasetFile	73
8.6.3.2 device_edgeMatrix	73
8.6.3.3 device_matrixIndex	73
8.6.3.4 device_totalEdgeWeights	73
8.6.3.5 device_valueMatrix	73
8.6.3.6 edgeMatrix	73
8.6.3.7 edges	74
8.6.3.8 matrixIndex	74
8.6.3.9 matrixSize	74
8.6.3.10 modelSavefile	74
8.6.3.11 nodes	74
8.6.3.12 outputFile	74
8.6.3.13 starterNode	74
8.6.3.14 totalEdgeWeights	74
8.6.3.15 valueMatrix	75
8.7 Markov::Random::DefaultRandomEngine Class Reference	75
8.7.1 Detailed Description	76
8.7.2 Member Function Documentation	77
8.7.2.1 distribution()	77
8.7.2.2 generator()	77
8.7.2.3 random()	78
8.7.2.4 rd()	78
8.8 Markov::Edge< NodeStorageType > Class Template Reference	79

8.8.1 Detailed Description	81
8.8.2 Constructor & Destructor Documentation	81
8.8.2.1 Edge() [1/2]	81
8.8.2.2 Edge() [2/2]	81
8.8.3 Member Function Documentation	81
8.8.3.1 AdjustEdge()	82
8.8.3.2 EdgeWeight()	82
8.8.3.3 LeftNode()	82
8.8.3.4 RightNode()	83
8.8.3.5 SetLeftEdge()	83
8.8.3.6 SetRightEdge()	83
8.8.3.7 TraverseNode()	84
8.8.4 Member Data Documentation	84
8.8.4.1 _left	84
8.8.4.2 _right	84
8.8.4.3 _weight	84
8.9 Markov::API::MarkovPasswords Class Reference	84
8.9.1 Detailed Description	87
8.9.2 Constructor & Destructor Documentation	88
8.9.2.1 MarkovPasswords() [1/2]	88
8.9.2.2 MarkovPasswords() [2/2]	88
8.9.3 Member Function Documentation	88
8.9.3.1 AdjustEdge()	88
8.9.3.2 Edges()	89
8.9.3.3 Export() [1/2]	89
8.9.3.4 Export() [2/2]	90
8.9.3.5 Generate()	90
8.9.3.6 GenerateThread()	91
8.9.3.7 Import() [1/2]	92
8.9.3.8 Import() [2/2]	92
8.9.3.9 Nodes()	93
8.9.3.10 OpenDatasetFile()	94
8.9.3.11 RandomWalk()	94
8.9.3.12 Save()	95
8.9.3.13 StarterNode()	96
8.9.3.14 Train()	96
8.9.3.15 TrainThread()	97
8.9.4 Member Data Documentation	98
8.9.4.1 datasetFile	98
8.9.4.2 edges	98
8.9.4.3 modelSavefile	98
8.9.4.4 nodes	98

8.9.4.5 outputFile	98
8.9.4.6 starterNode	99
8.10 Markov::GUI::MarkovPasswordsGUI Class Reference	99
8.10.1 Detailed Description	101
8.10.2 Constructor & Destructor Documentation	101
8.10.2.1 MarkovPasswordsGUI()	101
8.10.3 Member Function Documentation	101
8.10.3.1 loadDataset()	101
8.10.3.2 MarkovPasswordsGUI::benchmarkSelected	101
8.10.3.3 MarkovPasswordsGUI::comparisonSelected	101
8.10.3.4 MarkovPasswordsGUI::modelvisSelected	102
8.10.3.5 MarkovPasswordsGUI::visualDebugSelected	102
8.10.3.6 renderHTMLFile()	102
8.10.4 Member Data Documentation	102
8.10.4.1 ui	102
8.11 Markov::Random::Marsaglia Class Reference	102
8.11.1 Detailed Description	105
8.11.2 Constructor & Destructor Documentation	105
8.11.2.1 Marsaglia()	105
8.11.3 Member Function Documentation	105
8.11.3.1 distribution()	105
8.11.3.2 generator()	106
8.11.3.3 random()	106
8.11.3.4 rd()	107
8.11.4 Member Data Documentation	107
8.11.4.1 x	107
8.11.4.2 y	107
8.11.4.3 z	108
8.12 Markov::GUI::menu Class Reference	108
8.12.1 Detailed Description	109
8.12.2 Constructor & Destructor Documentation	109
8.12.2.1 menu()	109
8.12.3 Member Function Documentation	110
8.12.3.1 about	110
8.12.3.2 visualization	110
8.12.4 Member Data Documentation	110
8.12.4.1 ui	110
8.13 Markov::Random::Mersenne Class Reference	110
8.13.1 Detailed Description	112
8.13.2 Member Function Documentation	113
8.13.2.1 distribution()	113
8.13.2.2 generator()	113

8.13.2.3 random()	114
8.13.2.4 rd()	114
8.14 Markov::Model< NodeStorageType > Class Template Reference	115
8.14.1 Detailed Description	117
8.14.2 Constructor & Destructor Documentation	117
8.14.2.1 Model()	117
8.14.3 Member Function Documentation	117
8.14.3.1 AdjustEdge()	118
8.14.3.2 Edges()	118
8.14.3.3 Export() [1/2]	119
8.14.3.4 Export() [2/2]	119
8.14.3.5 Import() [1/2]	120
8.14.3.6 Import() [2/2]	120
8.14.3.7 Nodes()	122
8.14.3.8 RandomWalk()	122
8.14.3.9 StarterNode()	123
8.14.4 Member Data Documentation	124
8.14.4.1 edges	124
8.14.4.2 nodes	124
8.14.4.3 starterNode	124
8.15 Markov::API::ModelMatrix Class Reference	124
8.15.1 Detailed Description	128
8.15.2 Constructor & Destructor Documentation	128
8.15.2.1 ModelMatrix()	128
8.15.3 Member Function Documentation	128
8.15.3.1 AdjustEdge()	128
8.15.3.2 ConstructMatrix()	129
8.15.3.3 DumpJSON()	130
8.15.3.4 Edges()	131
8.15.3.5 Export() [1/2]	131
8.15.3.6 Export() [2/2]	132
8.15.3.7 FastRandomWalk()	132
8.15.3.8 FastRandomWalkPartition()	133
8.15.3.9 FastRandomWalkThread()	134
8.15.3.10 Generate()	136
8.15.3.11 GenerateThread()	137
8.15.3.12 Import() [1/2]	138
8.15.3.13 Import() [2/2]	138
8.15.3.14 Nodes()	139
8.15.3.15 OpenDatasetFile()	139
8.15.3.16 RandomWalk()	140
8.15.3.17 Save()	141

8.15.3.18 StarterNode()	142
8.15.3.19 Train()	142
8.15.3.20 TrainThread()	143
8.15.4 Member Data Documentation	144
8.15.4.1 datasetFile	144
8.15.4.2 edgeMatrix	144
8.15.4.3 edges	144
8.15.4.4 matrixIndex	144
8.15.4.5 matrixSize	144
8.15.4.6 modelSavefile	145
8.15.4.7 nodes	145
8.15.4.8 outputFile	145
8.15.4.9 starterNode	145
8.15.4.10 totalEdgeWeights	145
8.15.4.11 valueMatrix	145
8.16 Markov::Node< storageType > Class Template Reference	145
8.16.1 Detailed Description	148
8.16.2 Constructor & Destructor Documentation	148
8.16.2.1 Node() [1/2]	148
8.16.2.2 Node() [2/2]	148
8.16.3 Member Function Documentation	149
8.16.3.1 Edges()	149
8.16.3.2 FindEdge() [1/2]	149
8.16.3.3 FindEdge() [2/2]	149
8.16.3.4 Link() [1/2]	150
8.16.3.5 Link() [2/2]	150
8.16.3.6 NodeValue()	151
8.16.3.7 RandomNext()	151
8.16.3.8 TotalEdgeWeights()	152
8.16.3.9 UpdateEdges()	152
8.16.3.10 UpdateTotalVerticeWeight()	153
8.16.4 Member Data Documentation	153
8.16.4.1 _value	153
8.16.4.2 edges	153
8.16.4.3 edgesV	153
8.16.4.4 total_edge_weights	153
8.17 Markov::Random::RandomEngine Class Reference	154
8.17.1 Detailed Description	155
8.17.2 Member Function Documentation	155
8.17.2.1 random()	155
8.18 Markov::API::CLI::Terminal Class Reference	155
8.18.1 Detailed Description	157

8.18.2 Member Enumeration Documentation	157
8.18.2.1 color	157
8.18.3 Constructor & Destructor Documentation	157
8.18.3.1 Terminal()	157
8.18.4 Member Data Documentation	157
8.18.4.1 colormap	158
8.18.4.2 endl	158
8.19 Markov::API::Concurrency::ThreadSharedListHandler Class Reference	158
8.19.1 Detailed Description	160
8.19.2 Constructor & Destructor Documentation	160
8.19.2.1 ThreadSharedListHandler()	160
8.19.3 Member Function Documentation	161
8.19.3.1 next()	161
8.19.4 Member Data Documentation	161
8.19.4.1 listfile	161
8.19.4.2 mlock	161
8.20 Markov::GUI::Train Class Reference	162
8.20.1 Detailed Description	163
8.20.2 Constructor & Destructor Documentation	163
8.20.2.1 Train()	163
8.20.3 Member Function Documentation	163
8.20.3.1 home	163
8.20.3.2 train	163
8.20.4 Member Data Documentation	163
8.20.4.1 ui	163
9 File Documentation	165
9.1 about.h File Reference	165
9.2 about.h	165
9.3 argparse.cpp File Reference	166
9.4 argparse.cpp	166
9.5 argparse.h File Reference	166
9.5.1 Macro Definition Documentation	168
9.5.1.1 BOOST_ALL_DYN_LINK	168
9.6 argparse.h	168
9.7 CLI.h File Reference	170
9.8 CLI.h	171
9.9 cudaDeviceController.h File Reference	172
9.10 cudaDeviceController.h	172
9.11 cudaModelMatrix.h File Reference	173
9.12 cudaModelMatrix.h	174
9.13 dllmain.cpp File Reference	174

9.14 dllmain.cpp	175
9.15 edge.h File Reference	175
9.16 edge.h	176
9.17 framework.h File Reference	178
9.17.1 Macro Definition Documentation	179
9.17.1.1 WIN32_LEAN_AND_MEAN	179
9.18 framework.h	179
9.19 main.cpp File Reference	179
9.19.1 Function Documentation	179
9.19.1.1 main()	179
9.20 src/main.cpp	180
9.21 main.cpp File Reference	181
9.21.1 Function Documentation	181
9.21.1.1 main()	181
9.22 UI/src/main.cpp	182
9.23 markopy.cpp File Reference	182
9.23.1 Macro Definition Documentation	183
9.23.1.1 BOOST_PYTHON_STATIC_LIB	183
9.24 markopy.cpp	183
9.25 markopy_cli.py File Reference	184
9.26 markopy_cli.py	184
9.27 markovPasswords.cpp File Reference	187
9.28 markovPasswords.cpp	187
9.29 markovPasswords.h File Reference	189
9.30 markovPasswords.h	190
9.31 MarkovPasswordsGUI.cpp File Reference	191
9.32 MarkovPasswordsGUI.cpp	191
9.33 MarkovPasswordsGUI.h File Reference	192
9.34 MarkovPasswordsGUI.h	193
9.35 menu.cpp File Reference	194
9.36 menu.cpp	194
9.37 menu.h File Reference	195
9.38 menu.h	195
9.39 model.h File Reference	196
9.40 model.h	197
9.41 model_2gram.py File Reference	201
9.42 model_2gram.py	201
9.43 modelMatrix.cpp File Reference	202
9.44 modelMatrix.cpp	202
9.45 modelMatrix.h File Reference	205
9.46 modelMatrix.h	205
9.47 node.h File Reference	207

9.48 node.h	208
9.49 pch.cpp File Reference	211
9.50 MarkovModel/src/pch.cpp	212
9.51 pch.cpp File Reference	212
9.52 UnitTests/pch.cpp	212
9.53 pch.h File Reference	213
9.54 MarkovModel/src/pch.h	213
9.55 pch.h File Reference	214
9.56 UnitTests/pch.h	214
9.57 random-model.py File Reference	214
9.58 random-model.py	214
9.59 random.h File Reference	215
9.60 random.h	216
9.61 README.md File Reference	218
9.62 term.cpp File Reference	218
9.62.1 Function Documentation	219
9.62.1.1 operator<<()	219
9.63 term.cpp	219
9.64 term.h File Reference	220
9.64.1 Macro Definition Documentation	221
9.64.1.1 TERM_FAIL	221
9.64.1.2 TERM_INFO	222
9.64.1.3 TERM_SUCC	222
9.64.1.4 TERM_WARN	222
9.65 term.h	222
9.66 threadSharedListHandler.cpp File Reference	223
9.67 threadSharedListHandler.cpp	223
9.68 threadSharedListHandler.h File Reference	223
9.69 threadSharedListHandler.h	224
9.70 Train.h File Reference	226
9.71 Train.h	226
9.72 UnitTests.cpp File Reference	227
9.73 UnitTests.cpp	228
Index	237

Chapter 1

Markov Passwords

Table of Contents

Markov Passwords

Generate wordlists with markov models.

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

<details open="open">

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

</details>

1.1 About The Project

This project aims to generate wordlists using markov models.

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 PYTHON_PATH% (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 libboost 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/libboostpython38.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\um;C:\Program Files
(x86)\Windows Kits\10\include\10.0.19041.0\ucrt;C:\Program Files (x86)\Windows
Kits\10\include\10.0.19041.0\shared;C:\Program Files (x86)\Windows
Kits\10\include\10.0.19041.0\um;C:\Program Files (x86)\Windows
Kits\10\include\10.0.19041.0\winrt;C:\Program Files (x86)\Windows
Kits\10\include\10.0.19041.0\cppwinrt
set LIB=%LIB%;C:\Program Files (x86)\Windows Kits\10\lib\10.0.19041.0\ucrt\x64;C:\Program Files
(x86)\Windows Kits\10\lib\10.0.19041.0\um\x64
```

1.5.2.2 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 linguistic 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](#)

Chapter 2

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.

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

markopy_cli	17
Markov	
Namespace for the markov-model related classes. Contains Model , Node and Edge classes	20
Markov::API	
Namespace for the MarkovPasswords API	21
Markov::API::CLI	
Structure to hold parsed cli arguments	21
Markov::API::Concurrency	
Namespace for Concurrency related classes	22
Markov::API::CUDA	
Namespace for objects requiring CUDA libraries	22
Markov::GUI	
Namespace for MarkovPasswords API GUI wrapper	23
Markov::Markopy	23
Markov::Random	
Objects related to RNG	24
model_2gram	24
random	25
random-model	25
Testing	
Namespace for Microsoft Native Unit Testing Classes	25
Testing::MarkovModel	
Testing namespace for MarkovModel	26
Testing::MarkovPasswords	
Testing namespace for MarkovPasswords	29
Testing::MVP	
Testing Namespace for Minimal Viable Product	29
Testing::MVP::MarkovModel	
Testing Namespace for MVP MarkovModel	29
Testing::MVP::MarkovPasswords	
Testing namespace for MVP MarkovPasswords	35

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Markov::API::CLI::_programOptions	39
Markov::API::CLI::Argparse	44
Markov::API::CUDA::CUDADeviceController	51
Markov::API::CUDA::CUDAModelMatrix	53
Markov::Edge< NodeStorageType >	79
Markov::Edge< char >	79
Markov::Edge< storageType >	79
Markov::Model< NodeStorageType >	115
Markov::Model< char >	115
Markov::API::MarkovPasswords	84
Markov::API::ModelMatrix	124
Markov::API::CUDA::CUDAModelMatrix	53
Markov::Node< storageType >	145
Markov::Node< char >	145
Markov::Node< NodeStorageType >	145
QMainWindow	
Markov::GUI::about	42
Markov::GUI::CLI	49
Markov::GUI::MarkovPasswordsGUI	99
Markov::GUI::menu	108
Markov::GUI::Train	162
Markov::Random::RandomEngine	154
Markov::Random::DefaultRandomEngine	75
Markov::Random::Marsaglia	102
Markov::Random::Mersenne	110
Markov::API::CLI::Terminal	155
Markov::API::Concurrency::ThreadSharedListHandler	158

Chapter 5

Class Index

5.1 Class List

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

Markov::API::CLI::_programOptions	
Structure to hold parsed cli arguments	39
Markov::GUI::about	
QT Class for about page	42
Markov::API::CLI::Argparse	
Parse command line arguments	44
Markov::GUI::CLI	
QT CLI Class	49
Markov::API::CUDA::CUDADeviceController	
Controller class for CUDA device	51
Markov::API::CUDA::CUDAModelMatrix	
Extension of Markov::API::ModelMatrix which is modified to run on GPU devices	53
Markov::Random::DefaultRandomEngine	
Implementation using Random.h default random engine	75
Markov::Edge< NodeStorageType >	
Edge class used to link nodes in the model together	79
Markov::API::MarkovPasswords	
Markov::Model with char represented nodes	84
Markov::GUI::MarkovPasswordsGUI	
Reporting UI	99
Markov::Random::Marsaglia	
Implementation of Marsaglia Random Engine	102
Markov::GUI::menu	
QT Menu class	108
Markov::Random::Mersenne	
Implementation of Mersenne Twister Engine	110
Markov::Model< NodeStorageType >	
Class for the final Markov Model , constructed from nodes and edges	115
Markov::API::ModelMatrix	
Class to flatten and reduce Markov::Model to a Matrix	124
Markov::Node< storageType >	
A node class that for the vertices of model. Connected with eachother using Edge	145
Markov::Random::RandomEngine	
An abstract class for Random Engine	154
Markov::API::CLI::Terminal	
Pretty colors for Terminal . Windows Only	155
Markov::API::Concurrency::ThreadSharedListHandler	
Simple class for managing shared access to file	158
Markov::GUI::Train	
QT Training page class	162

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

about.h	165
argparse.cpp	166
argparse.h	166
CLI.h	170
cudaDeviceController.h	172
cudaModelMatrix.h	173
dllmain.cpp	174
edge.h	175
framework.h	178
src/main.cpp	179
UI/src/main.cpp	181
markopy.cpp	182
markopy_cli.py	184
markovPasswords.cpp	187
markovPasswords.h	189
MarkovPasswordsGUI.cpp	191
MarkovPasswordsGUI.h	192
menu.cpp	194
menu.h	195
model.h	196
model_2gram.py	201
modelMatrix.cpp	202
modelMatrix.h	205
node.h	207
MarkovModel/src/pch.cpp	211
UnitTests/pch.cpp	212
MarkovModel/src/pch.h	213
UnitTests/pch.h	214
random-model.py	214
random.h	215
term.cpp	218
term.h	220
threadSharedListHandler.cpp	223
threadSharedListHandler.h	223
Train.h	226
UnitTests.cpp	227

Chapter 7

Namespace Documentation

7.1 markopy_cli Namespace Reference

Functions

- def `cli_init` (input_model)
- def `cli_train` (model, dataset, seperator, output, output_forced=False, bulk=False)
- def `cli_generate` (model, wordlist, bulk=False)

Variables

- `parser`
- `help`
- `default`
- `action`
- `args` = parser.parse_args()
- `corpus_list` = os.listdir(args.dataset)
- def `model` = `cli_init`(args.input)
- `output_file_name` = corpus
- string `model_extension` = ""
- `output_forced`
- `True`
- `bulk`
- `model_list` = os.listdir(args.input)
- `model_base` = input
- `output`

7.1.1 Detailed Description

```
@namespace Markov::Markopy::Python
```

7.1.2 Function Documentation

7.1.2.1 cli_generate()

```
def markopy_cli.cli_generate (  
    model,  
    wordlist,  
    bulk = False )
```

Definition at line 114 of file [markopy_cli.py](#).

```
00114 def cli_generate(model, wordlist, bulk=False):
00115     if not (wordlist or args.count):
00116         logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters. Exiting.")
00117         exit(2)
00118
00119     if (bulk and os.path.isfile(wordlist)):
00120         logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00121     model.Generate(int(args.count), wordlist, int(args.min), int(args.max), int(args.threads))
00122
00123
```

7.1.2.2 cli_init()

```
def markopy_cli.cli_init (
    input_model )
```

Definition at line 61 of file [markopy_cli.py](#).

```
00061 def cli_init(input_model):
00062     logging.VERBOSITY = 0
00063     if args.verbosity:
00064         logging.VERBOSITY = args.verbosity
00065         logging.pprint(f"Verbosity set to {args.verbosity}.", 2)
00066
00067     logging.pprint("Initializing model.", 1)
00068     model = markopy.MarkovPasswords()
00069     logging.pprint("Model initialized.", 2)
00070
00071     logging.pprint("Importing model file.", 1)
00072
00073     if (not os.path.isfile(input_model)):
00074         logging.pprint(f"Model file at {input_model} not found. Check the file path, or working
00075         directory")
00076         exit(1)
00077
00078     model.Import(input_model)
00079     logging.pprint("Model imported successfully.", 2)
00080     return model
00081
```

7.1.2.3 cli_train()

```
def markopy_cli.cli_train (
    model,
    dataset,
    seperator,
    output,
    output_forced = False,
    bulk = False )
```

Definition at line 82 of file [markopy_cli.py](#).

```
00082 def cli_train(model, dataset, seperator, output, output_forced=False, bulk=False):
00083     if not (dataset and seperator and (output or not output_forced)):
00084         logging.pprint(
00085             f"Training mode requires -d/--dataset{' ', -o/--output' if output_forced else ''} and
-s/--seperator parameters. Exiting.")
00086         exit(2)
00087
00088     if (not bulk and not os.path.isfile(dataset)):
00089         logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00090         exit(3)
00091
00092     if (output and os.path.isfile(output)):
00093         logging.pprint(f"{output} exists and will be overwritten.", 1)
00094
00095     if (seperator == '\\t'):
00096         logging.pprint("Escaping seperator.", 3)
00097         seperator = '\t'
00098
00099     if (len(seperator) != 1):
00100         logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not accepted.')
00101         exit(4)
00102
00103     logging.pprint(f'Starting training.', 3)
00104     model.Train(dataset, seperator, int(args.threads))
00105     logging.pprint(f'Training completed.', 2)
00106
```

```
00107         if (output):
00108             logging.pprint(f'Exporting model to {output}', 2)
00109             model.Export(output)
00110         else:
00111             logging.pprint(f'Model will not be exported.', 1)
00112
00113
```

7.1.3 Variable Documentation

7.1.3.1 action

markopy_cli.action

Definition at line 49 of file [markopy_cli.py](#).

7.1.3.2 args

markopy_cli.args = parser.parse_args()

Definition at line 58 of file [markopy_cli.py](#).

7.1.3.3 bulk

markopy_cli.bulk

Definition at line 139 of file [markopy_cli.py](#).

7.1.3.4 corpus_list

markopy_cli.corpus_list = os.listdir(args.dataset)

Definition at line 130 of file [markopy_cli.py](#).

7.1.3.5 default

markopy_cli.default

Definition at line 41 of file [markopy_cli.py](#).

7.1.3.6 help

markopy_cli.help

Definition at line 27 of file [markopy_cli.py](#).

7.1.3.7 model

def markopy_cli.model = cli_init(args.input)

Definition at line 132 of file [markopy_cli.py](#).

7.1.3.8 model_base

markopy_cli.model_base = input

Definition at line 153 of file [markopy_cli.py](#).

7.1.3.9 model_extension

```
markopy_cli.model_extension = ""
```

Definition at line 135 of file [markopy_cli.py](#).

7.1.3.10 model_list

```
markopy_cli.model_list = os.listdir(args.input)
```

Definition at line 147 of file [markopy_cli.py](#).

7.1.3.11 output

```
markopy_cli.output
```

Definition at line 167 of file [markopy_cli.py](#).

7.1.3.12 output_file_name

```
markopy_cli.output_file_name = corpus
```

Definition at line 134 of file [markopy_cli.py](#).

7.1.3.13 output_forced

```
markopy_cli.output_forced
```

Definition at line 139 of file [markopy_cli.py](#).

7.1.3.14 parser

```
markopy_cli.parser
```

Initial value:

```
00001 = argparse.ArgumentParser(description="Python wrapper for MarkovPasswords.",
00002                               epilog=f, formatter_class=argparse.RawTextHelpFormatter)
```

Definition at line 12 of file [markopy_cli.py](#).

7.1.3.15 True

```
markopy_cli.True
```

Definition at line 139 of file [markopy_cli.py](#).

7.2 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](#).

7.2.1 Detailed Description

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

7.3 Markov::API Namespace Reference

Namespace for the [MarkovPasswords API](#).

Namespaces

- [CLI](#)
Structure to hold parsed cli arguments.
- [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.

7.3.1 Detailed Description

Namespace for the [MarkovPasswords API](#).

7.4 Markov::API::CLI Namespace Reference

Structure to hold parsed cli arguments.

Classes

- struct [_programOptions](#)
Structure to hold parsed cli arguments.
- class [Argparse](#)
Parse command line arguments.
- class [Terminal](#)
pretty colors for [Terminal](#). Windows Only.

Typedefs

- typedef struct [Markov::API::CLI::_programOptions](#) ProgramOptions
Structure to hold parsed cli arguments.

Functions

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

7.4.1 Detailed Description

Structure to hold parsed cli arguments.
Namespace for the CLI objects

7.4.2 Typedef Documentation

7.4.2.1 ProgramOptions

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

7.4.3 Function Documentation

7.4.3.1 operator<<()

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

overload for `std::cout`.

Definition at line 60 of file `term.cpp`.

```
00060                                     {
00061     char buf[6];
00062     sprintf(buf, "%d", Terminal::colormap.find(c)->second);
00063     os << "\e[1;" << buf << "m";
00064     return os;
00065 }
```

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

7.5 Markov::API::Concurrency Namespace Reference

Namespace for [Concurrency](#) related classes.

Classes

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

7.5.1 Detailed Description

Namespace for [Concurrency](#) related classes.

7.6 Markov::API::CUDA Namespace Reference

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

Classes

- class [CUDADeviceController](#)
Controller class for CUDA device.
- class [CUDAModelMatrix](#)

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

7.6.1 Detailed Description

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

7.7 Markov::GUI Namespace Reference

namespace for MarkovPasswords [API GUI](#) wrapper

Classes

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

7.7.1 Detailed Description

namespace for MarkovPasswords [API GUI](#) wrapper

7.8 Markov::Markopy Namespace Reference

Functions

- [BOOST_PYTHON_MODULE](#) (markopy)

7.8.1 Function Documentation

7.8.1.1 BOOST_PYTHON_MODULE()

Markov::Markopy::BOOST_PYTHON_MODULE (
 markopy)

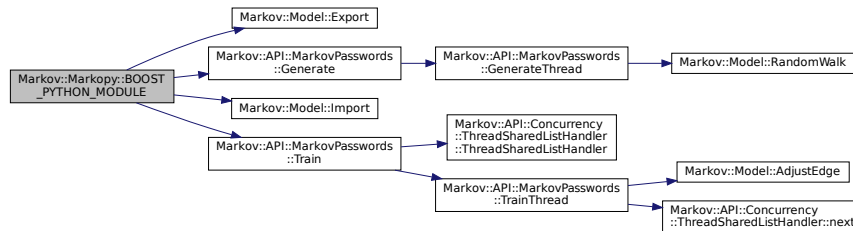
Definition at line 11 of file [markopy.cpp](#).

```
00012 {
00013     bool (Markov::API::MarkovPasswords::*Import)(const char*) = &Markov::Model<char>::Import;
00014     bool (Markov::API::MarkovPasswords::*Export)(const char*) = &Markov::Model<char>::Export;
00015     class_<Markov::API::MarkovPasswords>("MarkovPasswords", init<>())
00016         .def(init<>())
00017         .def("Train", &Markov::API::MarkovPasswords::Train,
00018             "Train the model\n"
00019             "\n"
00020             ":param datasetFileName: Ifstream* to the dataset. If null, use class member\n"
00021             ":param delimiter: a character, same as the delimiter in dataset content\n"
00022             ":param threads: number of OS threads to spawn\n")
00023         .def("Generate", &Markov::API::MarkovPasswords::Generate,
00024             "Generate passwords from a trained model.\n"
00025             ":param n: Ifstream* to the dataset. If null, use class member\n"
00026             ":param wordlistFileName: a character, same as the delimiter in dataset content\n"
00027             ":param minLen: number of OS threads to spawn\n"
00028             ":param maxLen: Ifstream* to the dataset. If null, use class member\n"
00029             ":param threads: a character, same as the delimiter in dataset content\n"
00030             ":param threads: number of OS threads to spawn\n")
00031         .def("Import", Import, "Import a model file.")
00032         .def("Export", Export, "Export a model to file.")
```

```
00033         ;
00034     };
```

References [Markov::Model< NodeStorageType >::Export\(\)](#), [Markov::API::MarkovPasswords::Generate\(\)](#), [Markov::Model< NodeStorageType >::Import\(\)](#) and [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the call graph for this function:



7.9 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](#).

7.9.1 Detailed Description

Objects related to RNG.

7.10 model_2gram Namespace Reference

Variables

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

7.10.1 Detailed Description

python script for generating a 2gram model

7.10.2 Variable Documentation

7.10.2.1 alphabet

`model_2gram.alphabet = string.printable`
password alphabet
Definition at line 10 of file [model_2gram.py](#).

7.10.2.2 f

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

7.11 random Namespace Reference

7.11.1 Detailed Description

-model

python script for generating a 2gram model

7.12 random-model Namespace Reference

Variables

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

7.12.1 Variable Documentation

7.12.1.1 alphabet

`random-model.alphabet = string.printable`
password alphabet
Definition at line 10 of file [random-model.py](#).

7.12.1.2 f

`random-model.f = open('../..models/random.mdl', "wb")`
output file handle
Definition at line 16 of file [random-model.py](#).

7.13 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.

7.13.1 Detailed Description

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

7.14 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.

7.14.1 Detailed Description

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

7.14.2 Function Documentation

7.14.2.1 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 492 of file [UnitTests.cpp](#).

```
00493     {
00494     public:
00497         TEST_METHOD(except_integer_underflow) {
00498             auto _underflow_adjust = [] {
00499                 Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00500                 Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00501                 Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
RightNode);
00502                 e->AdjustEdge(15);
00503                 e->AdjustEdge(-30);
00504                 delete LeftNode;
00505                 delete RightNode;
00506                 delete e;
00507             };
00508             Assert::ExpectException<std::underflow_error>(_underflow_adjust);
00509         }
00510
00513         TEST_METHOD(except_integer_overflow) {
00514             auto _overflow_adjust = [] {
00515                 Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00516                 Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
```

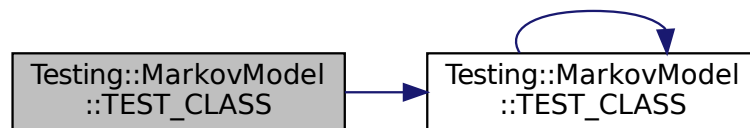
```

00517             Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00518             RightNode);
00518             e->AdjustEdge(~0ull);
00519             e->AdjustEdge(1);
00520             delete LeftNode;
00521             delete RightNode;
00522             delete e;
00523         };
00524         Assert::ExpectException<std::underflow_error>(_overflow_adjust);
00525     }
00526 };

```

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

Here is the call graph for this function:



7.14.2.2 TEST_CLASS() [2/3]

```

Testing::MarkovModel::TEST_CLASS (
    Model )

```

Test class for rest of model cases.

Definition at line 589 of file [UnitTests.cpp](#).

```

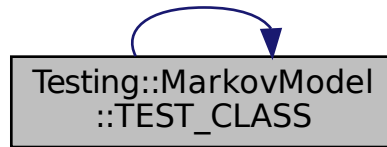
00590     {
00591     public:
00592         TEST_METHOD(functional_random_walk) {
00593             Markov::Model<unsigned char> m;
00594             Markov::Node<unsigned char>* starter = m.StarterNode();
00595             Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00596             Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
00597             Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00598             Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
00599             starter->Link(a)->AdjustEdge(1);
00600             a->Link(b)->AdjustEdge(1);
00601             b->Link(c)->AdjustEdge(1);
00602             c->Link(end)->AdjustEdge(1);
00603
00604             char* res = (char*)m.RandomWalk(1,12);
00605             Assert::IsFalse(strcmp(res, "abc"));
00606         }
00607         TEST_METHOD(functionoal_random_walk_without_any) {
00608             Markov::Model<unsigned char> m;
00609             Markov::Node<unsigned char>* starter = m.StarterNode();
00610             Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00611             Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
00612             Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00613             Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
00614             Markov::Edge<unsigned char>* res = NULL;
00615             starter->Link(a)->AdjustEdge(1);
00616             a->Link(b)->AdjustEdge(1);
00617             b->Link(c)->AdjustEdge(1);
00618             c->Link(end)->AdjustEdge(1);
00619
00620             res = starter->FindEdge('D');
00621             Assert::IsNull(res);
00622         }
00623     };
00624 };

```

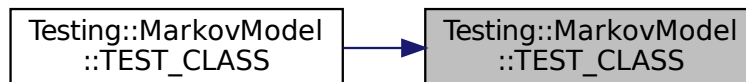
References [TEST_CLASS\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.2.3 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 530 of file [UnitTests.cpp](#).

```

00531     {
00532     public:
00533
00536         TEST_METHOD(rand_next_u64) {
00537
00538             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00539             Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00540             Markov::Edge<unsigned char>* e = src->Link(target1);
00541             e->AdjustEdge((unsigned long)(1ull << 63));
00542             Markov::Node<unsigned char>* res = src->RandomNext();
00543             Assert::IsTrue(res == target1);
00544             delete src;
00545             delete target1;
00546             delete e;
00547         }
00548
00549
00552         TEST_METHOD(rand_next_u64_max) {
00553
00554             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00555             Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00556             Markov::Edge<unsigned char>* e = src->Link(target1);
00557             e->AdjustEdge((0xffffffff));
00558             Markov::Node<unsigned char>* res = src->RandomNext();
00559             Assert::IsTrue(res == target1);
00560             delete src;
00561             delete target1;
00562             delete e;
00563
00564     }
  
```



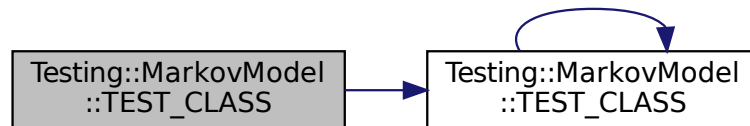
```

00565
00568     TEST_METHOD(uninitialized_rand_next) {
00569
00570         auto _invalid_next = [] {
00571             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00572             Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00573             Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(src, target1);
00574             Markov::Node<unsigned char>* res = src->RandomNext();
00575
00576             delete src;
00577             delete target1;
00578             delete e;
00579         };
00580
00581         Assert::ExpectException<std::logic_error>(_invalid_next);
00582     }
00583
00584
00585 };

```

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

Here is the call graph for this function:



7.15 Testing::MarkovPasswords Namespace Reference

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

7.15.1 Detailed Description

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

7.16 Testing::MVP Namespace Reference

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

Namespaces

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

7.16.1 Detailed Description

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

7.17 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.

7.17.1 Detailed Description

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

7.17.2 Function Documentation

7.17.2.1 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 21 of file [UnitTests.cpp](#).

```
00022     {
00023     public:
00024
00027         TEST_METHOD(default_constructor) {
00028             Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>;
00029             Assert::IsNull(e->LeftNode());
00030             Assert::IsNull(e->RightNode());
00031             delete e;
00032         }
00033
00036         TEST_METHOD(linked_constructor) {
00037             Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00038             Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00039             Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
RightNode);
00040             Assert::IsTrue(LeftNode == e->LeftNode());
00041             Assert::IsTrue(RightNode == e->RightNode());
00042             delete LeftNode;
00043             delete RightNode;
00044             delete e;
00045         }
00046
00049         TEST_METHOD(AdjustEdge) {
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 = new Markov::Edge<unsigned char>(LeftNode,
RightNode);
00053             e->AdjustEdge(15);
00054             Assert::AreEqual(15ull, e->EdgeWeight());
00055             e->AdjustEdge(15);
00056             Assert::AreEqual(30ull, e->EdgeWeight());
00057             delete LeftNode;
00058             delete RightNode;
00059             delete e;
00060         }
00061
00064         TEST_METHOD(TraverseNode) {
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 = new Markov::Edge<unsigned char>(LeftNode,
RightNode);
00068             Assert::IsTrue(RightNode == e->TraverseNode());
00069             delete LeftNode;
00070             delete RightNode;
```

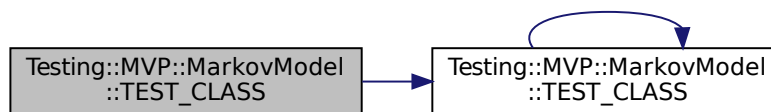
```

00071         delete e;
00072     }
00073
00074     TEST_METHOD(set_left_and_right) {
00075         Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00076         Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00077         Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(LeftNode,
00078             RightNode);
00079
00080         Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>;
00081         e2->SetLeftEdge(LeftNode);
00082         e2->SetRightEdge(RightNode);
00083
00084         Assert::IsTrue(e1->LeftNode() == e2->LeftNode());
00085         Assert::IsTrue(e1->RightNode() == e2->RightNode());
00086         delete LeftNode;
00087         delete RightNode;
00088         delete e1;
00089         delete e2;
00090     }
00091
00092     TEST_METHOD(negative_adjust) {
00093         Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00094         Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00095         Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00096             RightNode);
00097         e->AdjustEdge(15);
00098         Assert::AreEqual(15ull, e->EdgeWeight());
00099         e->AdjustEdge(-15);
00100         Assert::AreEqual(0ull, e->EdgeWeight());
00101         delete LeftNode;
00102         delete RightNode;
00103         delete e;
00104     }
00105 };

```

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

Here is the call graph for this function:



7.17.2.2 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 347 of file [UnitTests.cpp](#).

```

00348     {
00349     public:
00350         TEST_METHOD(model_constructor) {
00351             Markov::Model<unsigned char> m;
00352             Assert::AreEqual((unsigned char)'0', m.StarterNode()->NodeValue());
00353         }
00354
00355         TEST_METHOD(import_filename) {
00356             Markov::Model<unsigned char> m;
00357             Assert::IsTrue(m.Import("../MarkovPasswords/Models/2gram.mdl"));
00358         }
00359
00360         TEST_METHOD(export_filename) {
00361             Markov::Model<unsigned char> m;
00362             Assert::IsTrue(m.Export("../MarkovPasswords/Models/testcase.mdl"));
00363         }
00364     };

```

```

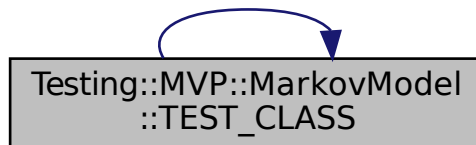
00369         }
00370
00373         TEST_METHOD(random_walk) {
00374             Markov::Model<unsigned char> m;
00375             Assert::IsTrue(m.Import("../models/finished.mdl"));
00376             Assert::IsNotNull(m.RandomWalk(1,12));
00377         }
00378     };

```

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.2.3 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 111 of file [UnitTests.cpp](#).

```

00112     {
00113     public:
00114
00117         TEST_METHOD(default_constructor) {
00118             Markov::Node<unsigned char>* n = new Markov::Node<unsigned char>();
00119             Assert::AreEqual((unsigned char)0, n->NodeValue());

```

```

00120         delete n;
00121     }
00122
00125     TEST_METHOD(uchar_constructor) {
00126         Markov::Node<unsigned char>* n = NULL;
00127         unsigned char test_cases[] = { 'c', 0x00, 0xff, -32 };
00128         for (unsigned char tcase : test_cases) {
00129             n = new Markov::Node<unsigned char>(tcase);
00130             Assert::AreEqual(tcase, n->NodeValue());
00131             delete n;
00132         }
00133     }
00134
00137     TEST_METHOD(link_left) {
00138         Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00139         Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00140
00141         Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
00142         delete LeftNode;
00143         delete RightNode;
00144         delete e;
00145     }
00146
00149     TEST_METHOD(link_right) {
00150         Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00151         Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00152
00153         Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(NULL, RightNode);
00154         LeftNode->Link(e);
00155         Assert::IsTrue(LeftNode == e->LeftNode());
00156         Assert::IsTrue(RightNode == e->RightNode());
00157         delete LeftNode;
00158         delete RightNode;
00159         delete e;
00160     }
00161
00164     TEST_METHOD(rand_next_low) {
00165
00166         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00167         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00168         Markov::Edge<unsigned char>* e = src->Link(target1);
00169         e->AdjustEdge(15);
00170         Markov::Node<unsigned char>* res = src->RandomNext();
00171         Assert::IsTrue(res == target1);
00172         delete src;
00173         delete target1;
00174         delete e;
00175     }
00176
00177
00180     TEST_METHOD(rand_next_u32) {
00181
00182         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00183         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00184         Markov::Edge<unsigned char>* e = src->Link(target1);
00185         e->AdjustEdge(1 << 31);
00186         Markov::Node<unsigned char>* res = src->RandomNext();
00187         Assert::IsTrue(res == target1);
00188         delete src;
00189         delete target1;
00190         delete e;
00191     }
00192
00193
00196     TEST_METHOD(rand_next_choice_1) {
00197
00198         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00199         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00200         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00201         Markov::Edge<unsigned char>* e1 = src->Link(target1);
00202         Markov::Edge<unsigned char>* e2 = src->Link(target2);
00203         e1->AdjustEdge(1);
00204         e2->AdjustEdge((unsigned long)(1ull << 31));
00205         Markov::Node<unsigned char>* res = src->RandomNext();
00206         Assert::IsNotNull(res);
00207         Assert::IsTrue(res == target2);
00208         delete src;
00209         delete target1;
00210         delete e1;
00211         delete e2;
00212     }
00213
00216     TEST_METHOD(rand_next_choice_2) {
00217
00218         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00219         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00220         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');

```

```

00221         Markov::Edge<unsigned char>* e1 = src->Link(target1);
00222         Markov::Edge<unsigned char>* e2 = src->Link(target2);
00223         e2->AdjustEdge(1);
00224         e1->AdjustEdge((unsigned long)(1ull « 31));
00225         Markov::Node<unsigned char>* res = src->RandomNext();
00226         Assert::IsNotNull(res);
00227         Assert::IsTrue(res == target1);
00228         delete src;
00229         delete target1;
00230         delete e1;
00231         delete e2;
00232     }
00233
00234
00237     TEST_METHOD(update_edges_count) {
00238
00239         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00240         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00241         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00242         Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00243         Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00244         e1->AdjustEdge(25);
00245         src->UpdateEdges(e1);
00246         e2->AdjustEdge(30);
00247         src->UpdateEdges(e2);
00248
00249         Assert::AreEqual((size_t)2, src->Edges()->size());
00250
00251         delete src;
00252         delete target1;
00253         delete e1;
00254         delete e2;
00255     }
00256
00257
00260     TEST_METHOD(update_edges_total) {
00261
00262         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00263         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00264         Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00265         Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target1);
00266         e1->AdjustEdge(25);
00267         src->UpdateEdges(e1);
00268         e2->AdjustEdge(30);
00269         src->UpdateEdges(e2);
00270
00271         Assert::AreEqual(55ull, src->TotalEdgeWeights());
00272
00273         delete src;
00274         delete target1;
00275         delete e1;
00276         delete e2;
00277     }
00278
00279
00282     TEST_METHOD(find_vertice) {
00283
00284         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00285         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00286         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00287         Markov::Edge<unsigned char>* res = NULL;
00288         src->Link(target1);
00289         src->Link(target2);
00290
00291
00292         res = src->FindEdge('b');
00293         Assert::IsNotNull(res);
00294         Assert::AreEqual((unsigned char)'b', res->TraverseNode()->NodeValue());
00295         res = src->FindEdge('c');
00296         Assert::IsNotNull(res);
00297         Assert::AreEqual((unsigned char)'c', res->TraverseNode()->NodeValue());
00298
00299         delete src;
00300         delete target1;
00301         delete target2;
00302
00303     }
00304
00305
00306
00309     TEST_METHOD(find_vertice_without_any) {
00310
00311         auto _invalid_next = [] {
00312             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00313             Markov::Edge<unsigned char>* res = NULL;
00314
00315             res = src->FindEdge('b');

```

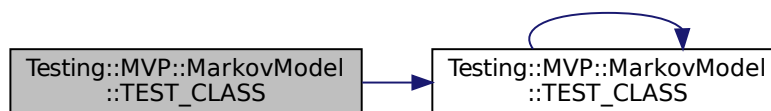
```

00316             Assert::IsNull(res);
00317
00318             delete src;
00319         };
00320
00321         //Assert::ExpectException<std::logic_error>(_invalid_next);
00322     }
00323
00326     TEST_METHOD(find_vertice_nonexistent) {
00327
00328         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00329         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00330         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00331         Markov::Edge<unsigned char>* res = NULL;
00332         src->Link(target1);
00333         src->Link(target2);
00334
00335         res = src->FindEdge('D');
00336         Assert::IsNull(res);
00337
00338         delete src;
00339         delete target1;
00340         delete target2;
00341     }
00342 }
00343 };

```

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

Here is the call graph for this function:



7.18 Testing::MVP::MarkovPasswords Namespace Reference

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

Functions

- [TEST_CLASS](#) (ArgParser)
Test Class for Argparse class.

7.18.1 Detailed Description

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

7.18.2 Function Documentation

7.18.2.1 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 generate

```

test basic generate

Definition at line 387 of file UnitTests.cpp.

```

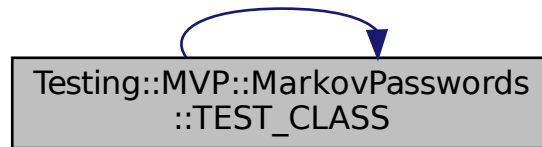
00388     {
00389     public:
00392         TEST_METHOD(generate_basic) {
00393             int argc = 8;
00394             char *argv[] = {"markov.exe", "generate", "-if", "model.mdl", "-of",
"passwords.txt", "-n", "100"};
00395
00396             /*ProgramOptions *p = Argparse::parse(argc, argv);
00397             Assert::IsNotNull(p);
00398
00399             Assert::AreEqual(p->bImport, true);
00400             Assert::AreEqual(p->bExport, false);
00401             Assert::AreEqual(p->importname, "model.mdl");
00402             Assert::AreEqual(p->outputfilename, "passwords.txt");
00403             Assert::AreEqual(p->generateN, 100); */
00404
00405         }
00406
00409         TEST_METHOD(generate_basic_reorder) {
00410             int argc = 8;
00411             char *argv[] = { "markov.exe", "generate", "-n", "100", "-if", "model.mdl", "-of",
"passwords.txt" };
00412
00413             /*ProgramOptions* p = Argparse::parse(argc, argv);
00414             Assert::IsNotNull(p);
00415
00416             Assert::AreEqual(p->bImport, true);
00417             Assert::AreEqual(p->bExport, false);
00418             Assert::AreEqual(p->importname, "model.mdl");
00419             Assert::AreEqual(p->outputfilename, "passwords.txt");
00420             Assert::AreEqual(p->generateN, 100); */
00421
00422         }
00425         TEST_METHOD(generate_basic_longname) {
00426             int argc = 8;
00427             char *argv[] = { "markov.exe", "generate", "-n", "100", "--inputfilename",
"model.mdl", "--outputfilename", "passwords.txt" };
00428
00429             /*ProgramOptions* p = Argparse::parse(argc, argv);
00430             Assert::IsNotNull(p);
00431
00432             Assert::AreEqual(p->bImport, true);
00433             Assert::AreEqual(p->bExport, false);
00434             Assert::AreEqual(p->importname, "model.mdl");
00435             Assert::AreEqual(p->outputfilename, "passwords.txt");
00436             Assert::AreEqual(p->generateN, 100); */
00437
00438         }
00441         TEST_METHOD(generate_fail_badmethod) {
00442             int argc = 8;
00443             char *argv[] = { "markov.exe", "junk", "-n", "100", "--inputfilename",
"model.mdl", "--outputfilename", "passwords.txt" };
00444
00445             /*ProgramOptions* p = Argparse::parse(argc, argv);
00446             Assert::IsNull(p); */
00447
00448         }
00451         TEST_METHOD(train_basic) {
00452             int argc = 4;
00453             char *argv[] = { "markov.exe", "train", "-ef", "model.mdl" };
00454
00455             /*ProgramOptions* p = Argparse::parse(argc, argv);
00456             Assert::IsNotNull(p);
00457
00458             Assert::AreEqual(p->bImport, false);
00459             Assert::AreEqual(p->bExport, true);
00460             Assert::AreEqual(p->exportname, "model.mdl"); */
00461
00462         }
00463
00466         TEST_METHOD(train_basic_longname) {
00467             int argc = 4;
00468             char *argv[] = { "markov.exe", "train", "--exportfilename", "model.mdl" };
00469
00470             /*ProgramOptions* p = Argparse::parse(argc, argv);
00471             Assert::IsNotNull(p);
00472
00473             Assert::AreEqual(p->bImport, false);
00474             Assert::AreEqual(p->bExport, true);
00475             Assert::AreEqual(p->exportname, "model.mdl"); */
00476
00477         }
00478
00479     };
00480

```

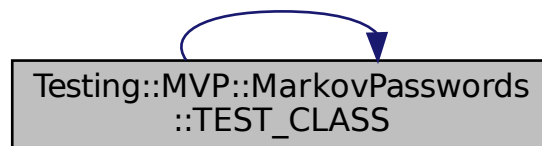

References [TEST_CLASS\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



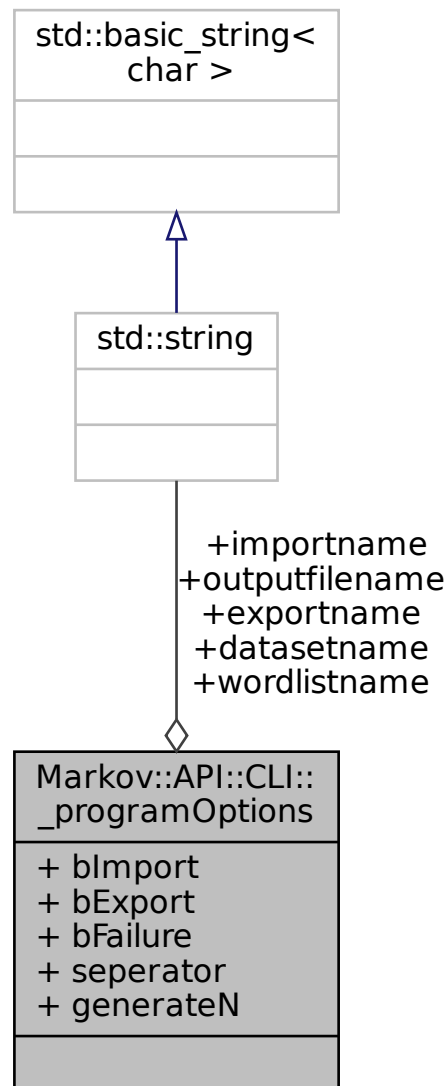
Chapter 8

Class Documentation

8.1 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 [blmport](#)
- bool [bExport](#)
- bool [bFailure](#)
- char [seperator](#)
- std::string [importname](#)
- std::string [exportname](#)
- std::string [wordlistname](#)
- std::string [outputfilename](#)
- std::string [datasetname](#)
- int [generateN](#)

8.1.1 Detailed Description

Structure to hold parsed cli arguments.
Definition at line 18 of file [argparse.h](#).

8.1.2 Member Data Documentation

8.1.2.1 bExport

`bool Markov::API::CLI::_programOptions::bExport`
Definition at line 20 of file [argparse.h](#).
Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

8.1.2.2 bFailure

`bool Markov::API::CLI::_programOptions::bFailure`
Definition at line 21 of file [argparse.h](#).
Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

8.1.2.3 bImport

`bool Markov::API::CLI::_programOptions::bImport`
Definition at line 19 of file [argparse.h](#).
Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

8.1.2.4 datasetname

`std::string Markov::API::CLI::_programOptions::datasetname`
Definition at line 27 of file [argparse.h](#).
Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

8.1.2.5 exportname

`std::string Markov::API::CLI::_programOptions::exportname`
Definition at line 24 of file [argparse.h](#).
Referenced by [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

8.1.2.6 generateN

`int Markov::API::CLI::_programOptions::generateN`
Definition at line 28 of file [argparse.h](#).
Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

8.1.2.7 importname

`std::string Markov::API::CLI::_programOptions::importname`
Definition at line 23 of file [argparse.h](#).
Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

8.1.2.8 outputfilename

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

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

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

8.1.2.9 seperator

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

Definition at line 22 of file [argparse.h](#).

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

8.1.2.10 wordlistname

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

Definition at line 25 of file [argparse.h](#).

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

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

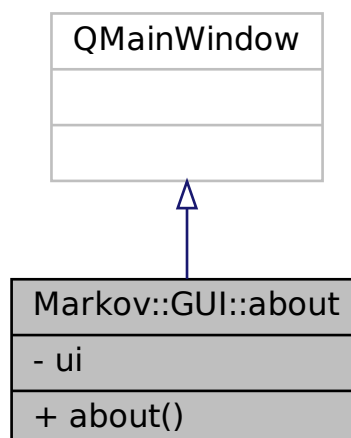
- [argparse.h](#)

8.2 Markov::GUI::about Class Reference

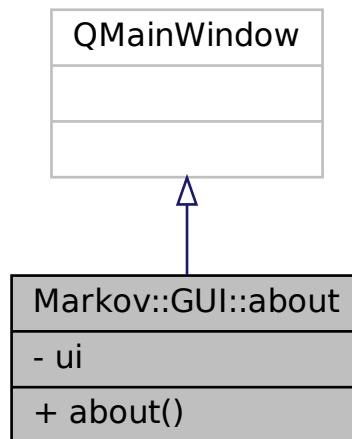
QT Class for about page.

```
#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](#)

8.2.1 Detailed Description

QT Class for about page.

Definition at line 12 of file [about.h](#).

8.2.2 Constructor & Destructor Documentation

8.2.2.1 about()

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

8.2.3 Member Data Documentation

8.2.3.1 ui

Ui:: [main](#) Markov::GUI::about::ui [private]

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

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

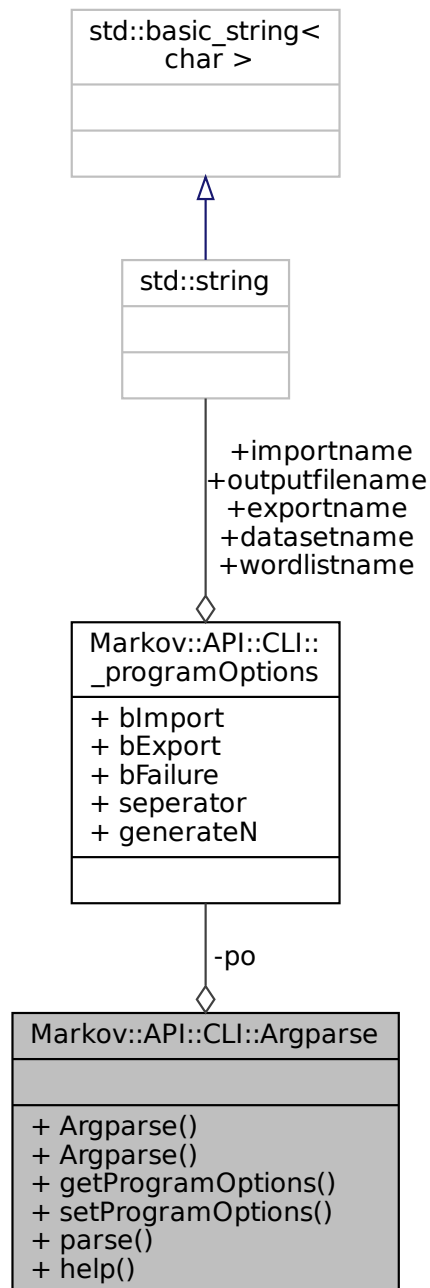
- [about.h](#)

8.3 Markov::API::CLI::Argparse Class Reference

Parse command line arguements.

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

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



Public Member Functions

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

Parse command line arguments.

- [Markov::API::CLI::ProgramOptions](#) `getProgramOptions` (void)

Getter for command line options.

- void [setProgramOptions](#) (bool i, bool e, bool bf, char s, std::string iName, std::string exName, std::string oName, std::string dName, int n)

Initialize program options structure.

Static Public Member Functions

- static [Markov::API::CLI::ProgramOptions](#) * `parse` (int argc, char **argv)

parse cli commands and return

- static void `help` ()

Print help string.

Private Attributes

- [Markov::API::CLI::ProgramOptions](#) `po`

8.3.1 Detailed Description

Parse command line arguments.

Definition at line 34 of file [argparse.h](#).

8.3.2 Constructor & Destructor Documentation

8.3.2.1 Argparse() [1/2]

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

8.3.2.2 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

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

Definition at line 46 of file [argparse.h](#).

```
00046                                     {
00047
00048         /*bool bImp;
00049         bool bExp;
00050         bool bFail;
00051         char sptr;
00052         std::string imports;
00053         std::string exports;
00054         std::string outputs;
00055         std::string datasets;
00056         int generateN;
00057         */
00058         opt::options_description desc("Options");
00059
00060
```

```

00061         desc.add_options()
00062             ("generate", "Generate strings with given parameters")
00063             ("train", "Train model with given parameters")
00064             ("combine", "Combine")
00065             ("import", opt::value<std::string>(), "Import model file")
00066             ("output", opt::value<std::string>(), "Output model file. This model will be exported
when done. Will be ignored for generation mode")
00067             ("dataset", opt::value<std::string>(), "Dataset file to read input from training. Will
be ignored for generation mode")
00068             ("separator", opt::value<char>(), "Separator character to use with training data.
(character between occurrence and value)")
00069             ("wordlist", opt::value<std::string>(), "Wordlist file path to export generation
results to. Will be ignored for training mode")
00070             ("count", opt::value<int>(), "Number of lines to generate. Ignored in training mode")
00071             ("verbosity", "Output verbosity")
00072             ("help", "Option definitions");
00073
00074         opt::variables_map vm;
00075
00076         opt::store(opt::parse_command_line(argc, argv, desc), vm);
00077
00078         opt::notify(vm);
00079
00080         //std::cout << desc << std::endl;
00081         if (vm.count("help")) {
00082             std::cout << desc << std::endl;
00083         }
00084
00085         if (vm.count("output") == 0) this->po.outputfilename = "NULL";
00086         else if (vm.count("output") == 1) {
00087             this->po.outputfilename = vm["output"].as<std::string>();
00088             this->po.bExport = true;
00089         }
00090         else {
00091             this->po.bFailure = true;
00092             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00093             std::cout << desc << std::endl;
00094         }
00095
00096
00097         if (vm.count("dataset") == 0) this->po.datasetname = "NULL";
00098         else if (vm.count("dataset") == 1) {
00099             this->po.datasetname = vm["dataset"].as<std::string>();
00100         }
00101         else {
00102             this->po.bFailure = true;
00103             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00104             std::cout << desc << std::endl;
00105         }
00106
00107
00108         if (vm.count("wordlist") == 0) this->po.wordlistname = "NULL";
00109         else if (vm.count("wordlist") == 1) {
00110             this->po.wordlistname = vm["wordlist"].as<std::string>();
00111         }
00112         else {
00113             this->po.bFailure = true;
00114             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00115             std::cout << desc << std::endl;
00116         }
00117
00118         if (vm.count("import") == 0) this->po.importname = "NULL";
00119         else if (vm.count("import") == 1) {
00120             this->po.importname = vm["import"].as<std::string>();
00121             this->po.bImport = true;
00122         }
00123         else {
00124             this->po.bFailure = true;
00125             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00126             std::cout << desc << std::endl;
00127         }
00128
00129
00130         if (vm.count("count") == 0) this->po.generateN = 0;
00131         else if (vm.count("count") == 1) {
00132             this->po.generateN = vm["count"].as<int>();
00133         }
00134         else {
00135             this->po.bFailure = true;
00136             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00137             std::cout << desc << std::endl;
00138         }
00139
00140         /*std::cout << vm["output"].as<std::string>() << std::endl;
00141         std::cout << vm["dataset"].as<std::string>() << std::endl;
00142         std::cout << vm["wordlist"].as<std::string>() << std::endl;
00143         std::cout << vm["output"].as<std::string>() << std::endl;

```

```

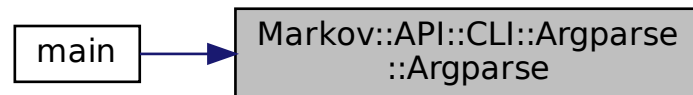
00144         std::cout << vm["count"].as<int>() << std::endl;*/
00145
00146
00147         //else if (vm.count("train")) std::cout << "train oldu" << std::endl;
00148     }

```

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:



8.3.3 Member Function Documentation

8.3.3.1 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 155 of file [argparse.h](#).

```

00155     {
00156         return this->po;
00157     }

```

References [po](#).

8.3.3.2 help()

```

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

```

Print help string.

Definition at line 8 of file [argparse.cpp](#).

```

00008     {
00009         std::cout <<
00010         "Markov Passwords - Help\n"
00011         "Options:\n"
00012         "  \n"
00013         "  -of --outputfilename\n"
00014         "      Filename to output the generation results\n"
00015         "  -ef --exportfilename\n"
00016         "      filename to export built model to\n"
00017         "  -if --importfilename\n"
00018         "      filename to import model from\n"
00019         "  -n (generate count)\n"
00020         "      Number of lines to generate\n"
00021         "  \n"
00022         "Usage: \n"
00023         "  markov.exe -if empty_model.mdl -ef model.mdl\n"
00024         "      import empty_model.mdl and train it with data from stdin. When done, output the model to
00025         model.mdl\n"
00025         "  \n"

```

```

00026      "    markov.exe -if empty_model.mdl -n 15000 -of wordlist.txt\n"
00027      "        import empty_model.mdl and generate 15000 words to wordlist.txt\n"
00028
00029      « std::endl;
00030 }

```

8.3.3.3 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 arguement count
<i>argv</i>	- Program arguement values array

Returns

ProgramOptions structure.

Definition at line 4 of file [argparse.cpp](#).

```
00004 { return 0; }
```

8.3.3.4 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>	seperator 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 172 of file [argparse.h](#).

```

00172
00173      this->po.bImport = i;
00174      this->po.bExport = e;
00175      this->po.seperator = s;

```

```

00176         this->po.bFailure = bf;
00177         this->po.generateN = n;
00178         this->po.importname = iName;
00179         this->po.exportname = exName;
00180         this->po.outputfilename = oName;
00181         this->po.datasetname = dName;
00182
00183         /*strcpy_s(this->po.importname,256,iName);
00184         strcpy_s(this->po.exportname,256,exName);
00185         strcpy_s(this->po.outputfilename,256,oName);
00186         strcpy_s(this->po.datasetname,256,dName);*/
00187
00188     }

```

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

8.3.4 Member Data Documentation

8.3.4.1 po

[Markov::API::CLI::ProgramOptions](#) [Markov::API::CLI::Argparse::po](#) [private]

Definition at line 203 of file [argparse.h](#).

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

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

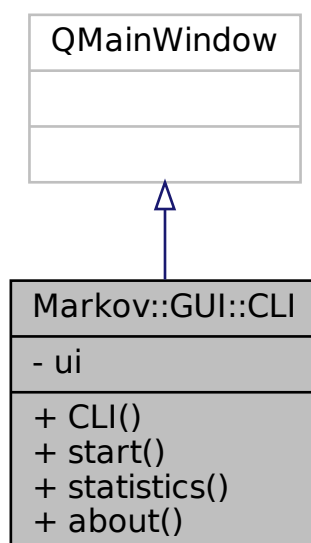
- [argparse.h](#)
- [argparse.cpp](#)

8.4 Markov::GUI::CLI Class Reference

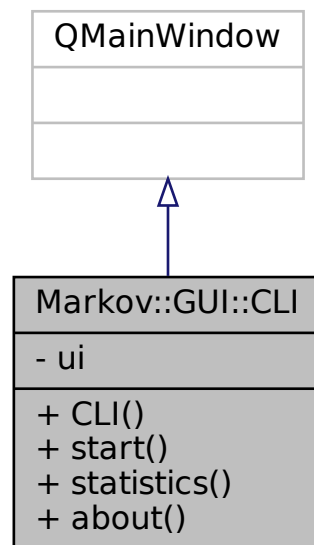
QT [CLI](#) Class.

```
#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](#)

8.4.1 Detailed Description

QT [CLI](#) Class.

Definition at line 8 of file [CLI.h](#).

8.4.2 Constructor & Destructor Documentation

8.4.2.1 CLI()

```

Markov::GUI::CLI::CLI (
    QWidget * parent = Q_NULLPTR )
  
```

8.4.3 Member Function Documentation

8.4.3.1 about

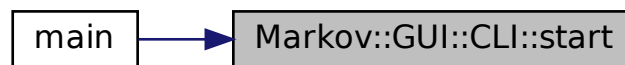
```
void Markov::GUI::CLI::about ( ) [slot]
```

8.4.3.2 start

```
void Markov::GUI::CLI::start ( ) [slot]
```

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

Here is the caller graph for this function:



8.4.3.3 statistics

```
void Markov::GUI::CLI::statistics ( ) [slot]
```

8.4.4 Member Data Documentation

8.4.4.1 ui

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

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

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

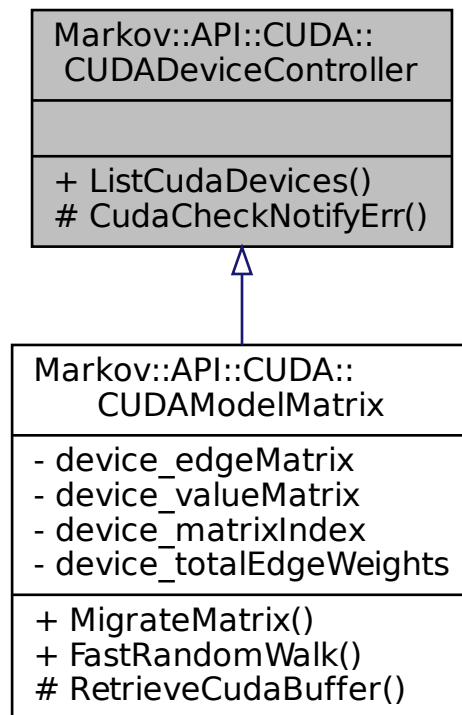
- [CLI.h](#)

8.5 Markov::API::CUDA::CUDADeviceController Class Reference

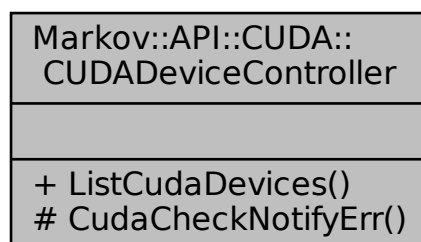
Controller class for [CUDA](#) device.

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

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



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



Public Member Functions

- `__host__ void ListCudaDevices ()`

*List **CUDA** devices in the system.*

Protected Member Functions

- `__host__ int CudaCheckNotifyErr(cudaError_t _status, const char *msg)`
Check results of the last operation on GPU.

8.5.1 Detailed Description

Controller class for [CUDA](#) device.

This implementation only supports Nvidia devices.

Definition at line 9 of file [cudaDeviceController.h](#).

8.5.2 Member Function Documentation

8.5.2.1 CudaCheckNotifyErr()

```
__host__ int Markov::API::CUDA::CUDADeviceController::CudaCheckNotifyErr (
    cudaError_t _status,
    const char * msg ) [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");
```

8.5.2.2 ListCudaDevices()

```
__host__ void Markov::API::CUDA::CUDADeviceController::ListCudaDevices ( )
```

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
```

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

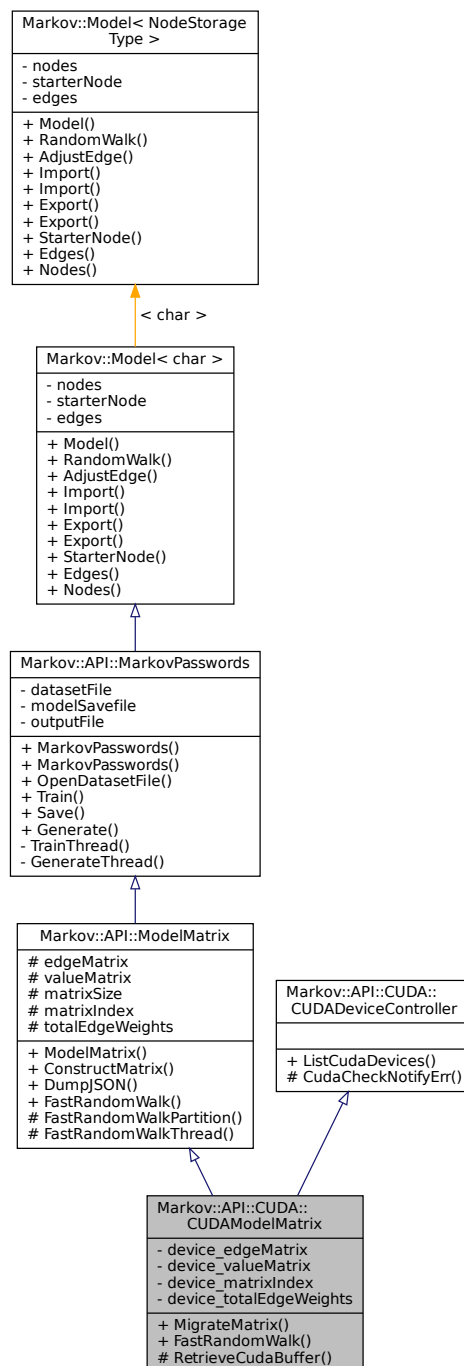
- [cudaDeviceController.h](#)

8.6 Markov::API::CUDA::CUDAModelMatrix Class Reference

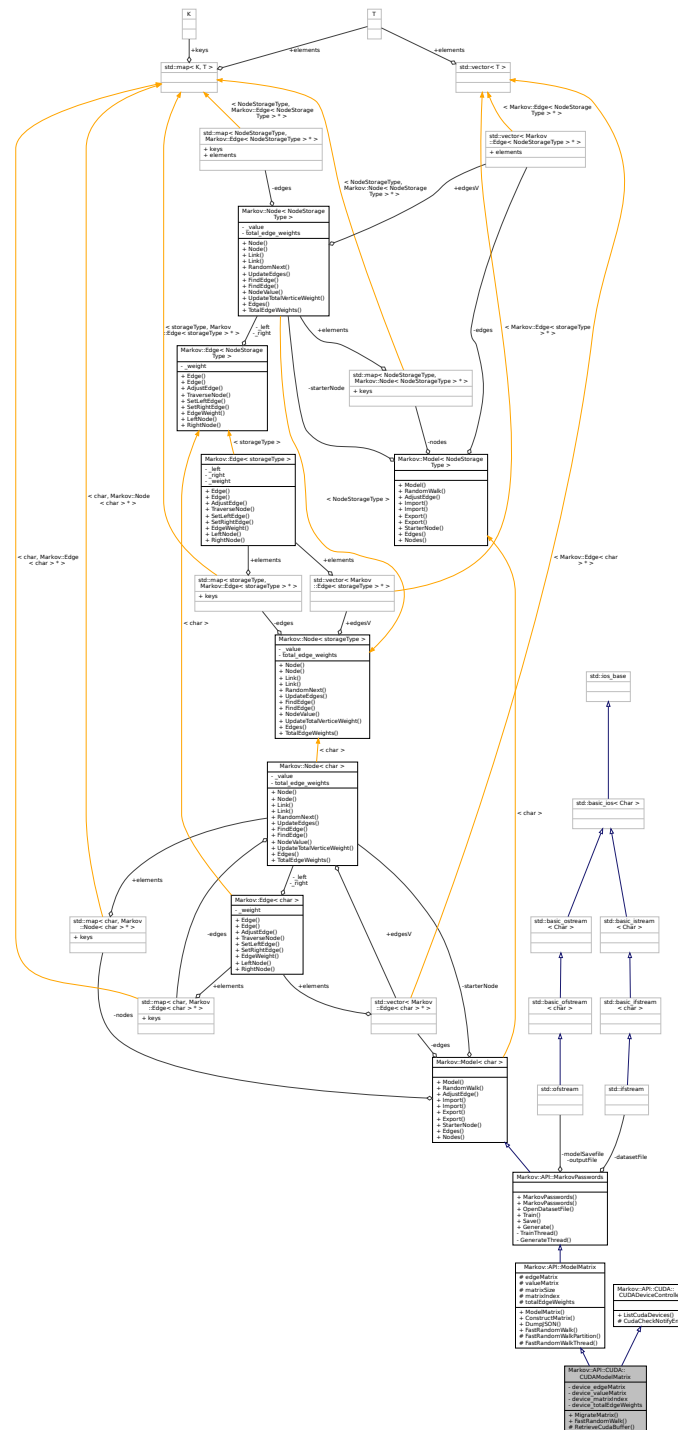
Extension of [Markov::API::ModelMatrix](#) which is modified to run on GPU devices.

```
#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 FastRandomWalk (unsigned long int n, const char *wordlistFileName, int minLen, int maxLen, int threads, bool bFileIO)`
[Random](#) walk on the Matrix-reduced [Markov::Model](#).
- `void ConstructMatrix ()`

- *Construct the related Matrix data for the model.*
- void [DumpJSON](#) ()
Debug function to dump the model to a JSON 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.
- 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.
- __host__ void [ListCudaDevices](#) ()
List [CUDA](#) devices in the system.

Protected Member Functions

- __host__ void [RetrieveCudaBuffer](#) ()
Retrieve the result buffer from [CUDA](#) kernel.
- 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](#).
- __host__ int [CudaCheckNotifyErr](#) (cudaError_t _status, const char *msg)
Check results of the last operation on GPU.

Protected Attributes

- char ** [edgeMatrix](#)
- long int ** [valueMatrix](#)
- int [matrixSize](#)
- char * [matrixIndex](#)
- long int * [totalEdgeWeights](#)

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](#)
- long int ** [device_valueMatrix](#)
- char * [device_matrixIndex](#)
- long int * [device_totalEdgeWeights](#)
- std::ifstream * [datasetFile](#)
- std::ofstream * [modelSavefile](#)
- std::ofstream * [outputFile](#)
- 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.

8.6.1 Detailed Description

Extension of [Markov::API::ModelMatrix](#) which is modified to run on GPU devices.
This implementation only supports Nvidia devices.
Definition at line 11 of file [cudaModelMatrix.h](#).

8.6.2 Member Function Documentation

8.6.2.1 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

<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 322 of file [model.h](#).

```
00322
00323     NodeStorageType p = payload[0];
```

```
{
```

```

00324     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00325     Markov::Edge<NodeStorageType>* e;
00326     int i = 0;
00327
00328     if (p == 0) return;
00329     while (p != 0) {
00330         e = curnode->FindEdge(p);
00331         if (e == NULL) return;
00332         e->AdjustEdge(occurrence);
00333         curnode = e->RightNode();
00334         p = payload[++i];
00335     }
00336
00337     e = curnode->FindEdge('\xff');
00338     e->AdjustEdge(occurrence);
00339     return;
00340 }

```

8.6.2.2 ConstructMatrix()

void 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.

Definition at line 11 of file modelMatrix.cpp.

```

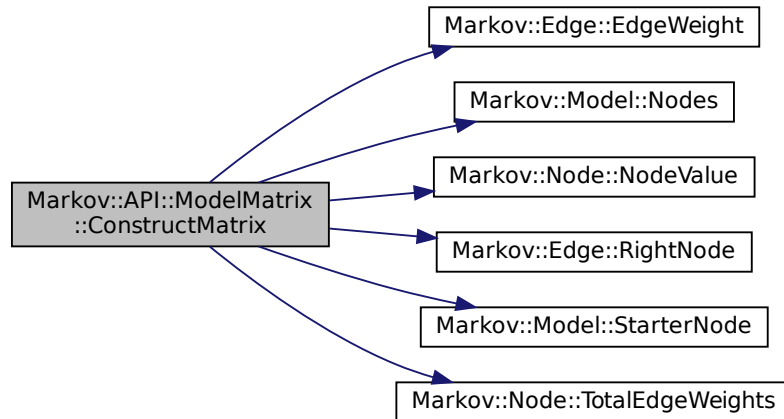
00011     {
00012         this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00013
00014         this->matrixIndex = new char[this->matrixSize];
00015         this->totalEdgeWeights = new long int[this->matrixSize];
00016
00017         this->edgeMatrix = new char*[this->matrixSize];
00018         for(int i=0;i<this->matrixSize;i++){
00019             this->edgeMatrix[i] = new char[this->matrixSize];
00020         }
00021         this->valueMatrix = new long int*[this->matrixSize];
00022         for(int i=0;i<this->matrixSize;i++){
00023             this->valueMatrix[i] = new long int[this->matrixSize];
00024         }
00025         std::map< char, Node< char > * > *nodes;
00026         nodes = this->Nodes();
00027         int i=0;
00028         for (auto const& [repr, node] : *nodes){
00029             if(repr!=0) this->matrixIndex[i] = repr;
00030             else this->matrixIndex[i] = 199;
00031             this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00032             for(int j=0;j<this->matrixSize;j++){
00033                 char val = node->NodeValue();
00034                 if(val < 0){
00035                     for(int k=0;k<this->matrixSize;k++){
00036                         this->valueMatrix[i][k] = 0;
00037                         this->edgeMatrix[i][k] = 255;
00038                     }
00039                     break;
00040                 }
00041                 else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00042                     this->valueMatrix[i][j] = 0;
00043                     this->edgeMatrix[i][j] = 255;
00044                 }else if(j==(this->matrixSize-1)) {
00045                     this->valueMatrix[i][j] = 0;
00046                     this->edgeMatrix[i][j] = 255;
00047                 }else{
00048                     this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00049                     this->edgeMatrix[i][j] = node->edgesV[j]->RightNode()->NodeValue();
00050                 }
00051             }
00052             i++;
00053         }
00054     }
00055
00056     //this->DumpJSON();
00057 }

```

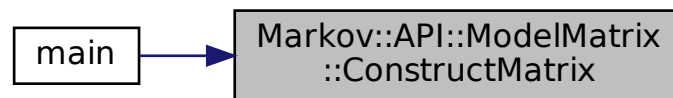
References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), [Markov::Node< storageType >::NodeValue\(\)](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#),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

[Markov::Node< storageType >::TotalEdgeWeights\(\)](#), and [Markov::API::ModelMatrix::valueMatrix](#).
Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.6.2.3 CudaCheckNotifyErr()

```

__host__ int Markov::API::CUDA::CUDADeviceController::CudaCheckNotifyErr (
    cudaError_t _status,
    const char * msg ) [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");
  
```

8.6.2.4 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 60 of file [modelMatrix.cpp](#).

```

00060         {
00061
00062         std::cout << "{\n  \"index\": \n";
00063         for(int i=0;i<this->matrixSize;i++){
00064             if(this->matrixIndex[i]=='') std::cout << "\\\"";
00065             else if(this->matrixIndex[i]=='\\') std::cout << "\\\\";
00066             else if(this->matrixIndex[i]==0) std::cout << "\\x00";
00067             else if(i==0) std::cout << "\\xff";
00068             else if(this->matrixIndex[i]=='\\n') std::cout << "\\n";
00069             else std::cout << this->matrixIndex[i];
00070         }
00071         std::cout <<
00072         "\\\", \n"
00073         "  \"edgemap\": {\n";
00074
00075         for(int i=0;i<this->matrixSize;i++){
00076             if(this->matrixIndex[i]=='') std::cout << "    \"\\\"\": [";
00077             else if(this->matrixIndex[i]=='\\') std::cout << "    \"\\\\\"\": [";
00078             else if(this->matrixIndex[i]==0) std::cout << "    \"\\x00\"\": [";
00079             else if(this->matrixIndex[i]<0) std::cout << "    \"\\xff\"\": [";
00080             else std::cout << "    \"\" < this->matrixIndex[i] << \"\"\": [";
00081             for(int j=0;j<this->matrixSize;j++){
00082                 if(this->edgeMatrix[i][j]=='') std::cout << "\"\\\"";
00083                 else if(this->edgeMatrix[i][j]=='\\') std::cout << "\"\\\\\"";
00084                 else if(this->edgeMatrix[i][j]==0) std::cout << "\"\\x00\"";
00085                 else if(this->edgeMatrix[i][j]<0) std::cout << "\"\\xff\"";
00086                 else if(this->matrixIndex[i]=='\\n') std::cout << "\"\\n\"";
00087                 else std::cout << "\"\" < this->edgeMatrix[i][j] << "\"\"";
00088                 if(j!=this->matrixSize-1) std::cout << ", ";
00089             }
00090             std::cout << "], \n";
00091         }
00092         std::cout << "}, \n";
00093
00094         std::cout << "\"  \"weightmap\": {\n";
00095         for(int i=0;i<this->matrixSize;i++){
00096             if(this->matrixIndex[i]=='') std::cout << "    \"\\\"\": [";
00097             else if(this->matrixIndex[i]=='\\') std::cout << "    \"\\\\\"\": [";
00098             else if(this->matrixIndex[i]==0) std::cout << "    \"\\x00\"\": [";
00099             else if(this->matrixIndex[i]<0) std::cout << "    \"\\xff\"\": [";
00100             else std::cout << "    \"\" < this->matrixIndex[i] << \"\"\": [";
00101             for(int j=0;j<this->matrixSize;j++){
00102                 std::cout << this->valueMatrix[i][j];
00103                 if(j!=this->matrixSize-1) std::cout << ", ";
00104             }
00105             std::cout << "], \n";
00106         }
00107         std::cout << " } \n\n";
00109     }

```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), and [Markov::API::ModelMatrix::valueMatrix](#).

8.6.2.5 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 172 of file [model.h](#).

```
00172 { return &edges; }
```

8.6.2.6 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 285 of file [model.h](#).

```
00285                                     {
00286     std::ofstream exportfile;
00287     exportfile.open(filename);
00288     return this->Export(&exportfile);
00289 }
```

8.6.2.7 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 273 of file [model.h](#).

```
00273                                     {
00274     Markov::Edge<NodeStorageType>* e;
00275     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00276         e = this->edges[i];
00277         //std::cout << e->LeftNode()->NodeValue() << ", " << e->EdgeWeight() << ", " <<
e->RightNode()->NodeValue() << "\n";
00278         *f << e->LeftNode()->NodeValue() << ", " << e->EdgeWeight() << ", " << e->RightNode()->NodeValue() <<
"\n";
00279     }
00280     return true;
00281 }
00282 }
```

8.6.2.8 FastRandomWalk()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen,
    int maxLen,
    int threads,
    bool bFileIO )
```

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

Parameters

<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);
```

8.6.2.9 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.

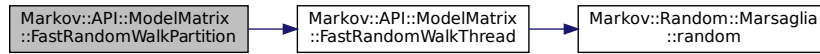
Definition at line 182 of file [modelMatrix.cpp](#).

```
00182                                     {
00183
00184     int iterationsPerThread = n/threads;
00185     int iterationsPerThreadCarryOver = n%threads;
00186
00187     std::vector<std::thread*> threadsV;
00188
00189     int id = 0;
00190     for(int i=0;i<threads;i++){
00191         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00192             mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00193         id++;
00194     }
00195     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00196         wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00197
00198     for(int i=0;i<threads;i++){
00199         threadsV[i]->join();
00200     }
```

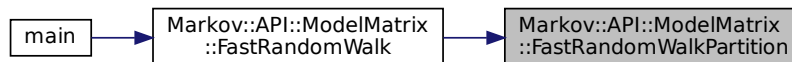
References [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.6.2.10 FastRandomWalkThread()

```

void Markov::API::ModelMatrix::FastRandomWalkThread (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    int id,
    bool bFileIO ) [protected], [inherited]
  
```

A single thread of a single partition of FastRandomWalk.

A FastRandomWalkPartition will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>id</i>	- DEPRECATED Thread id - No longer used
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 112 of file [modelMatrix.cpp](#).

```

00112
00113         if (n==0) return;
00114
00115         Markov::Random::Marsaglia MarsagliaRandomEngine;
00116         char* e;
00117         char *res = new char[maxLen*n];
00118         int index = 0;
00119         char next;
00120         int len=0;
00121         long int selection;
00122         char cur;
  
```

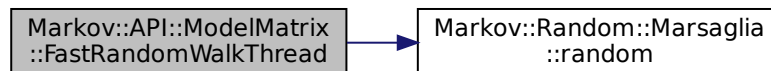
```

00123     long int bufferctr = 0;
00124     for (int i = 0; i < n; i++) {
00125         cur=199;
00126         len=0;
00127         while (true) {
00128             e = strchr(this->matrixIndex, cur);
00129             index = e - this->matrixIndex;
00130             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00131             for(int j=0;j<this->matrixSize;j++){
00132                 selection -= this->valueMatrix[index][j];
00133                 if (selection < 0){
00134                     next = this->edgeMatrix[index][j];
00135                     break;
00136                 }
00137             }
00138             if (len >= maxLen) break;
00139             else if ((next < 0) && (len < minLen)) continue;
00140             else if (next < 0) break;
00141             cur = next;
00142             res[bufferctr + len++] = cur;
00143         }
00144         res[bufferctr + len++] = '\n';
00145         bufferctr+=len;
00146     }
00147 }
00148 if(bFileIO){
00149     mlock->lock();
00150     *wordlist « res;
00151     mlock->unlock();
00152 }else{
00153     mlock->lock();
00154     std::cout « res;
00155     mlock->unlock();
00156 }
00157 delete res;
00158 }
00159 }
00160 }

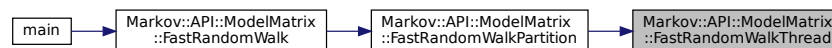
```

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:



8.6.2.11 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 92 of file [markovPasswords.cpp](#).

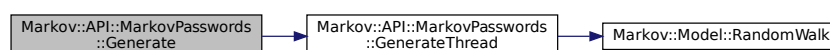
```

00092
{
00093     char* res;
00094     char print[100];
00095     std::ofstream wordlist;
00096     wordlist.open(wordlistFileName);
00097     std::mutex mlock;
00098     int iterationsPerThread = n/threads;
00099     int iterationsCarryOver = n%threads;
00100     std::vector<std::thread*> threadsV;
00101     for(int i=0;i<threads;i++){
00102         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
&mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00103     }
00104
00105     for(int i=0;i<threads;i++){
00106         threadsV[i]->join();
00107         delete threadsV[i];
00108     }
00109
00110     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00111
00112 }
```

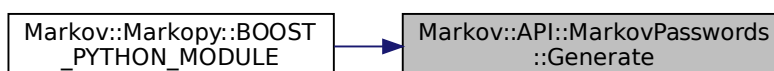
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.6.2.12 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 114 of file `markovPasswords.cpp`.

```

00114
00115     {
00116     char* res = new char[maxLen+5];
00117     if(n==0) return;
00118     Markov::Random::Marsaglia MarsagliaRandomEngine;
00119     for (int i = 0; i < n; i++) {
00120         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00121         outputLock->lock();
00122         *wordlist « res « "\n";
00123         outputLock->unlock();
00124     }
00125 }

```

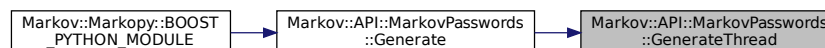
References `Markov::Model< NodeStorageType >::RandomWalk()`.

Referenced by `Markov::API::MarkovPasswords::Generate()`.

Here is the call graph for this function:



Here is the caller graph for this function:



8.6.2.13 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 265 of file [model.h](#).

```
00265                                     {
00266     std::ifstream importfile;
00267     importfile.open(filename);
00268     return this->Import(&importfile);
00269 }
00270 }
```

8.6.2.14 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 206 of file [model.h](#).

```
00206                                     {
00207     std::string cell;
00208
00209     char src;
00210     char target;
00211     long int oc;
00212
00213     while (std::getline(*f, cell)) {
00214         //std::cout << "cell: " << cell << std::endl;
00215         src = cell[0];
00216         target = cell[cell.length() - 1];
00217         char* j;
00218         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00219         //std::cout << oc << "\n";
00220         Markov::Node<NodeStorageType>* srcN;
00221         Markov::Node<NodeStorageType>* targetN;
00222         Markov::Edge<NodeStorageType>* e;
00223         if (this->nodes.find(src) == this->nodes.end()) {
00224             srcN = new Markov::Node<NodeStorageType>(src);
00225             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00226             //std::cout << "Creating new node at start.\n";
00227         }
00228         else {
00229             srcN = this->nodes.find(src)->second;
00230         }
00231
00232         if (this->nodes.find(target) == this->nodes.end()) {
00233             targetN = new Markov::Node<NodeStorageType>(target);
00234             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00235             //std::cout << "Creating new node at end.\n";
00236         }
00237         else {
00238             targetN = this->nodes.find(target)->second;
00239         }
00240         e = srcN->Link(targetN);
00241         e->AdjustEdge(oc);
00242         this->edges.push_back(e);
00243
00244         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00245         int(targetN->NodeValue()) << "\n";
00246     }
00247 }
00248
00249 for (std::pair<unsigned char, Markov::Node<NodeStorageType>> const& x : this->nodes) {
00250     //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00251 }
```

```

00251         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [] (Edge<NodeStorageType> *lhs,
Edge<NodeStorageType> *rhs)->bool{
00252             return lhs->EdgeWeight() > rhs->EdgeWeight();
00253         });
00254         //for(int i=0;i<x.second->edgesV.size();i++)
00255         // std::cout << x.second->edgesV[i]->EdgeWeight() << " ";
00256         //std::cout << "\n";
00257     }
00258     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00259     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00260
00261     return true;
00262 }

```

8.6.2.15 ListCudaDevices()

`__host__ void Markov::API::CUDA::CUDADeviceController::ListCudaDevices ()` [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

```

8.6.2.16 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.

8.6.2.17 Nodes()

`std::map<char , Node<char >*> Markov::Model< char >::Nodes ()` [inline], [inherited]

Return starter [Node](#).

Returns

starter node with 00 NodeValue

Definition at line 177 of file [model.h](#).

```
00177 { return &nodes; }
```

8.6.2.18 OpenDatasetFile()

`std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (`
`const char * filename)` [inherited]

Open dataset file and return the ifstream pointer.

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

ifstream* to the the dataset file

Definition at line 27 of file [markovPasswords.cpp](#).

```

00027
00028
00029     std::ifstream* datasetFile;

```



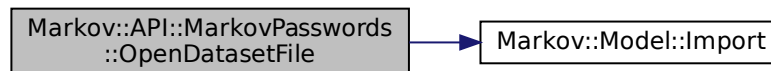
```

00030
00031     std::ifstream newFile(filename);
00032
00033     datasetFile = &newFile;
00034
00035     this->Import(datasetFile);
00036     return datasetFile;
00037 }

```

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

Here is the call graph for this function:



8.6.2.19 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 criateria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```

Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}

```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 292 of file [model.h](#).

```

00292
00293         {
00294             Markov::Node<NodeStorageType>* n = this->starterNode;
00295             int len = 0;
00296             Markov::Node<NodeStorageType>* temp_node;
00297             while (true) {
00298                 temp_node = n->RandomNext(randomEngine);
00299                 if (len >= maxSetting) {
00300                     break;
00301                 }
00302                 else if ((temp_node == NULL) && (len < minSetting)) {
00303                     continue;
00304                 }
00305                 else if (temp_node == NULL) {
00306                     break;
00307                 }
00308             }
00309             n = temp_node;
00310             buffer[len++] = n->NodeValue();
00311         }
00312         //null terminate the string
00313         buffer[len] = 0x00;
00314         //do something with the generated string
00315         return buffer; //for now
00316     }
00317 }

```

8.6.2.20 RetrieveCudaBuffer()

__host__ void Markov::API::CUDA::CUDAModelMatrix::RetrieveCudaBuffer () [protected]

Retrieve the result buffer from [CUDA](#) kernel.

Done on each partition.

8.6.2.21 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 80 of file [markovPasswords.cpp](#).

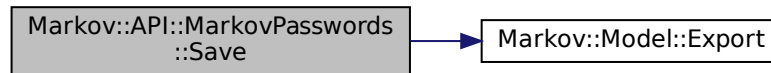
```

00080
00081         std::ofstream* exportFile;
00082
00083         std::ofstream newFile(filename);
00084
00085         exportFile = &newFile;
00086
00087         this->Export(exportFile);
00088         return exportFile;
00089     }

```

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

Here is the call graph for this function:



8.6.2.22 StarterNode()

`Node<char>* Markov::Model<char>::StarterNode () [inline], [inherited]`

Return starter [Node](#).

Returns

starter node with 00 NodeValue

Definition at line 167 of file [model.h](#).

```
00167 { return starterNode; }
```

8.6.2.23 Train()

```
void Markov::API::MarkovPasswords::Train (
    const char * datasetFileName,
    char delimiter,
    int threads ) [inherited]
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import ("models/2gram.mdl");
mp.Train("password.corpus");
```

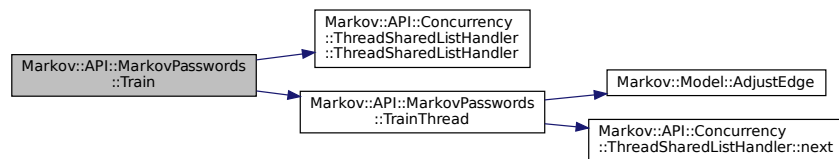
Definition at line 40 of file [markovPasswords.cpp](#).

```
00040
00041     Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
00042     auto start = std::chrono::high_resolution_clock::now();
00043
00044     std::vector<std::thread*> threadsV;
00045     for(int i=0;i<threads;i++){
00046         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
00047             &listhandler, 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 }
```

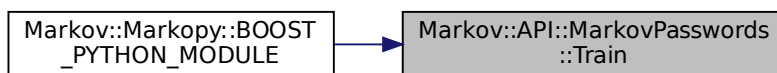
References [Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler\(\)](#), and [Markov::API::MarkovPasswords::](#)

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.6.2.24 TrainThread()

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

A single thread invoked by the Train function.

Parameters

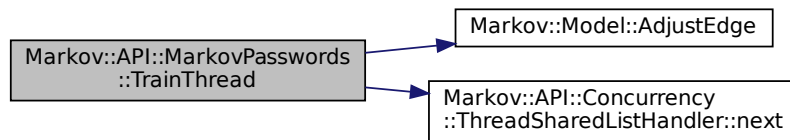
<i>listhandler</i>	- Listhandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

Definition at line 59 of file [markovPasswords.cpp](#).

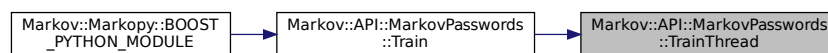
```
00059
{
00060     char format_str[] = "%ld,%s";
00061     format_str[2]=delimiter;
00062     std::string line;
00063     while (listhandler->next(&line)) {
00064         long int oc;
00065         if (line.size() > 100) {
00066             line = line.substr(0, 100);
00067         }
00068         char* linebuf = new char[line.length()+5];
00069 #ifdef _WIN32
00070         sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
00071 #else
00072         sscanf(line.c_str(), format_str, &oc, linebuf);
00073 #endif
00074         this->AdjustEdge((const char*)linebuf, oc);
00075         delete linebuf;
00076     }
00077 }
```

References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), 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:



8.6.3 Member Data Documentation

8.6.3.1 datasetFile

`std::ifstream*` `Markov::API::MarkovPasswords::datasetFile` [private], [inherited]
 Definition at line 106 of file [markovPasswords.h](#).

8.6.3.2 device_edgeMatrix

`char**` `Markov::API::CUDA::CUDAModelMatrix::device_edgeMatrix` [private]
 Definition at line 57 of file [cudaModelMatrix.h](#).

8.6.3.3 device_matrixIndex

`char*` `Markov::API::CUDA::CUDAModelMatrix::device_matrixIndex` [private]
 Definition at line 59 of file [cudaModelMatrix.h](#).

8.6.3.4 device_totalEdgeWeights

`long int*` `Markov::API::CUDA::CUDAModelMatrix::device_totalEdgeWeights` [private]
 Definition at line 60 of file [cudaModelMatrix.h](#).

8.6.3.5 device_valueMatrix

`long int**` `Markov::API::CUDA::CUDAModelMatrix::device_valueMatrix` [private]
 Definition at line 58 of file [cudaModelMatrix.h](#).

8.6.3.6 edgeMatrix

`char**` `Markov::API::ModelMatrix::edgeMatrix` [protected], [inherited]
 Definition at line 112 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

8.6.3.7 edges

```
std::vector<Edge<char >*> Markov::Model< char >::edges [private], [inherited]
```

A list of all edges in this model.

Definition at line 194 of file [model.h](#).

8.6.3.8 matrixIndex

```
char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]
```

Definition at line 115 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

8.6.3.9 matrixSize

```
int Markov::API::ModelMatrix::matrixSize [protected], [inherited]
```

Definition at line 114 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

8.6.3.10 modelSavefile

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

Definition at line 107 of file [markovPasswords.h](#).

8.6.3.11 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 183 of file [model.h](#).

8.6.3.12 outputFile

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

Definition at line 108 of file [markovPasswords.h](#).

8.6.3.13 starterNode

```
Node<char >* Markov::Model< char >::starterNode [private], [inherited]
```

Starter [Node](#) of this model.

Definition at line 188 of file [model.h](#).

8.6.3.14 totalEdgeWeights

```
long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]
```

Definition at line 116 of file [modelMatrix.h](#).

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

8.6.3.15 valueMatrix

long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]

Definition at line 113 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

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

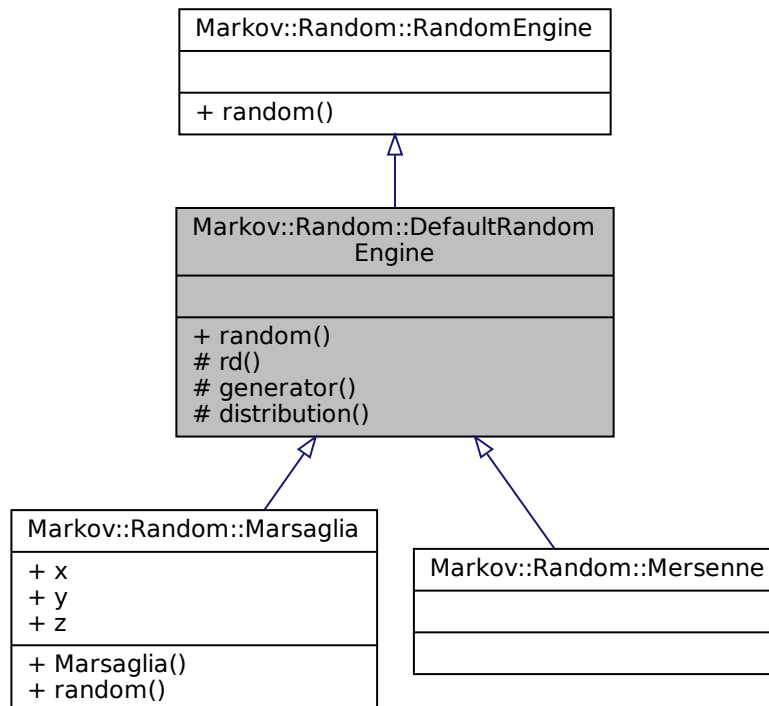
- [cudaModelMatrix.h](#)

8.7 Markov::Random::DefaultRandomEngine Class Reference

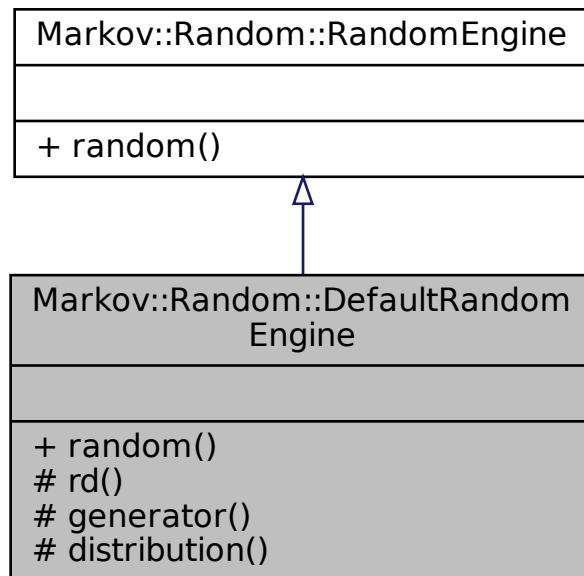
Implementation using [Random.h](#) default random engine.

```
#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.

8.7.1 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 [52](#) of file [random.h](#).

8.7.2 Member Function Documentation

8.7.2.1 distribution()

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

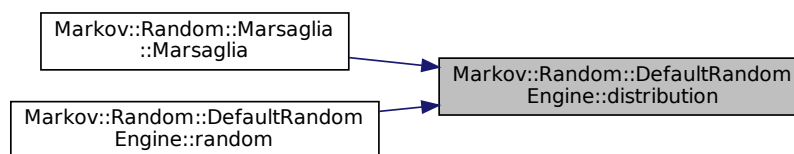
Distribution schema for seeding.

Definition at line 81 of file [random.h](#).

```
00081                                     {
00082         static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffffff);
00083         return _distribution;
00084     }
```

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

Here is the caller graph for this function:



8.7.2.2 generator()

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

Default random engine for seeding.

Definition at line 73 of file [random.h](#).

```
00073                                     {
00074         static std::default_random_engine _generator(rd());
00075         return _generator;
00076     }
```

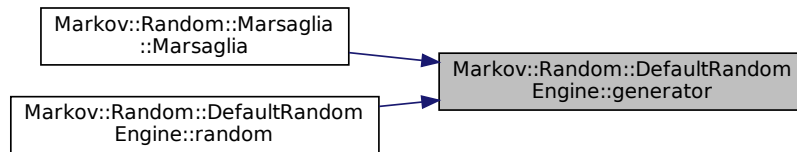
References [rd\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.7.2.3 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](#).

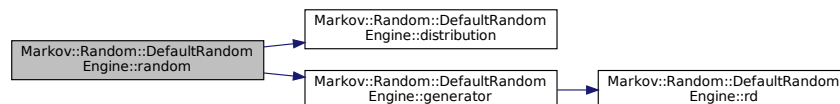
Definition at line 57 of file [random.h](#).

```

00057     {
00058         return this->distribution() (this->generator());
00059     }
  
```

References [distribution\(\)](#), and [generator\(\)](#).

Here is the call graph for this function:



8.7.2.4 rd()

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

Default random device for seeding.

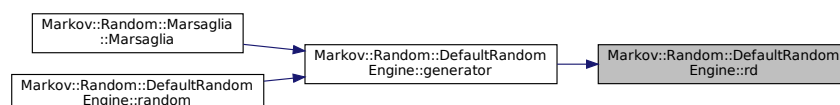
Definition at line 65 of file [random.h](#).

```

00065     {
00066         static std::random_device _rd;
00067         return _rd;
00068     }
  
```

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

Here is the caller graph for this function:



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

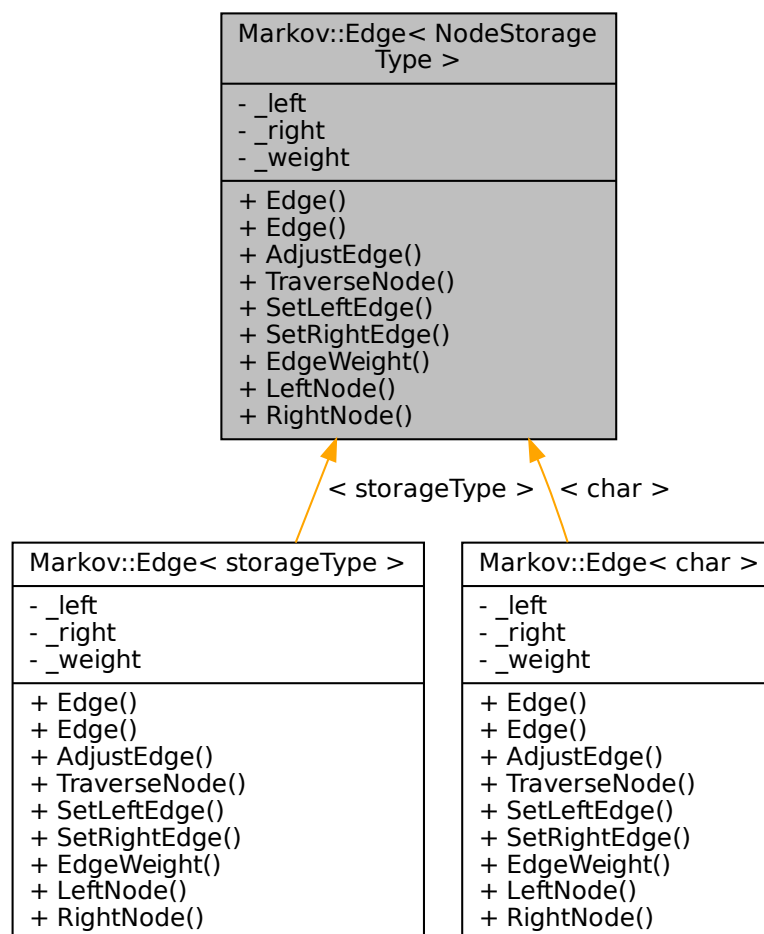
- [random.h](#)

8.8 Markov::Edge< NodeStorageType > Class Template Reference

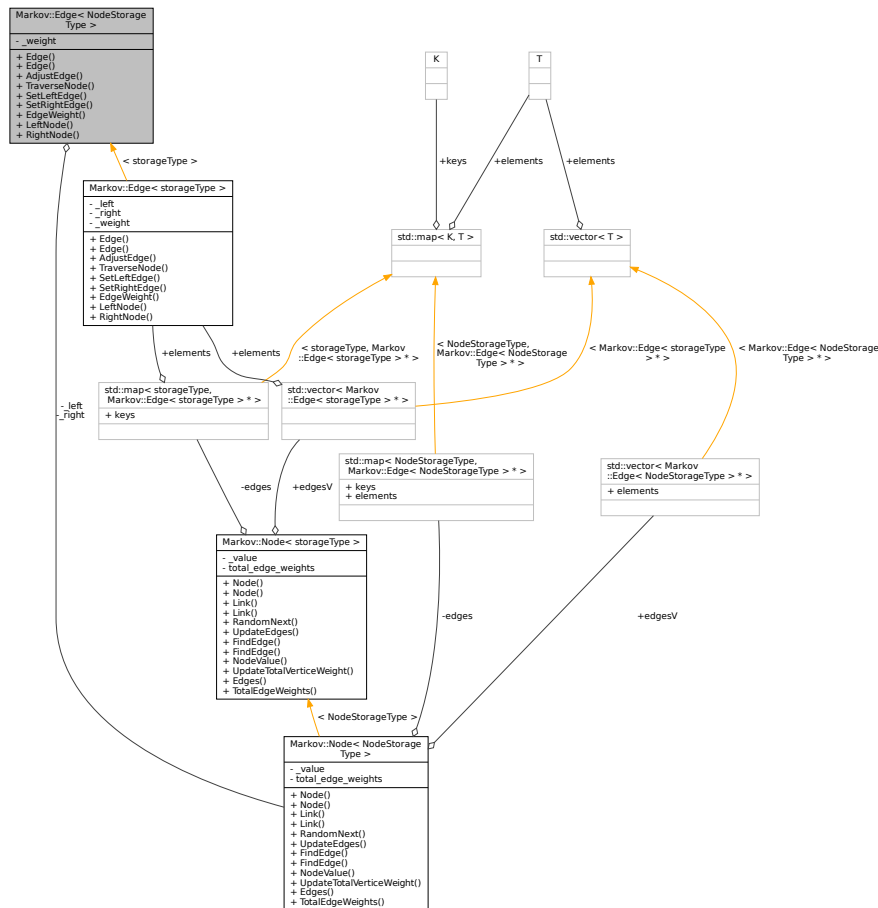
[Edge](#) class used to link nodes in the model together.

```
#include <model.h>
```

Inheritance diagram for Markov::Edge< NodeStorageType >:



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](#)
- [Node](#)< NodeStorageType > * [_right](#)
source node
- long int [_weight](#)
target node

8.8.1 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 26 of file [model.h](#).

8.8.2 Constructor & Destructor Documentation

8.8.2.1 Edge() [1/2]

```
template<typename NodeStorageType >
Markov::Edge< NodeStorageType >::Edge
```

Default constructor.

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

```
00105 {
00106     this->_left = NULL;
00107     this->_right = NULL;
00108     this->_weight = 0;
00109 }
```

8.8.2.2 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

_left	- Left node of this edge.
_right	- 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 112 of file [edge.h](#).

```
00112 {
00113     this->_left = _left;
00114     this->_right = _right;
00115     this->_weight = 0;
00116 }
```

8.8.3 Member Function Documentation

8.8.3.1 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 119 of file [edge.h](#).

```
00119                                     {
00120     this->_weight += offset;
00121     this->LeftNode()->UpdateTotalVerticeWeight(offset);
00122 }
```

8.8.3.2 EdgeWeight()

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

Returns

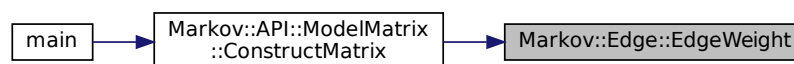
edge's EdgeWeight.

Definition at line 142 of file [edge.h](#).

```
00142                                     {
00143     return this->_weight;
00144 }
```

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

Here is the caller graph for this function:



8.8.3.3 LeftNode()

```
template<typename NodeStorageType >
Markov::Node< NodeStorageType > * Markov::Edge< NodeStorageType >::LeftNode
return edge's LeftNode
```

Returns

edge's LeftNode.

Definition at line 147 of file [edge.h](#).

```
00147                                     {
00148     return this->_left;
00149 }
```

8.8.3.4 RightNode()

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

Returns

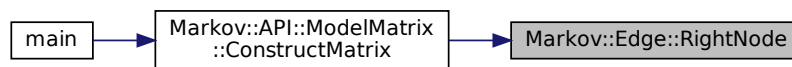
edge's RightNode.

Definition at line 152 of file [edge.h](#).

```
00152                                     {
00153     return this->_right;
00154 }
```

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

Here is the caller graph for this function:



8.8.3.5 SetLeftEdge()

```
template<typename NodeStorageType >
void Markov::Edge< NodeStorageType >::SetLeftEdge (
    Markov::Node< NodeStorageType > * n )
```

Set LeftNode of this edge.

Parameters

<i>node</i>	- Node to be linked with.
-------------	---

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

```
00132                                     {
00133     this->_left = n;
00134 }
```

8.8.3.6 SetRightEdge()

```
template<typename NodeStorageType >
void Markov::Edge< NodeStorageType >::SetRightEdge (
    Markov::Node< NodeStorageType > * n )
```

Set RightNode of this edge.

Parameters

<i>node</i>	- Node to be linked with.
-------------	---

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

```
00137                                     {
00138     this->_right = n;
00139 }
```

8.8.3.7 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 125 of file [edge.h](#).

```
00125
00126     if (this->RightNode()->NodeValue() == 0xff) //terminator node
00127         return NULL;
00128     return _right;
00129 }
```

8.8.4 Member Data Documentation

8.8.4.1 _left

```
template<typename NodeStorageType >
```

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

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

8.8.4.2 _right

```
template<typename NodeStorageType >
```

```
Node<NodeStorageType>* Markov::Edge< NodeStorageType >::_right [private]
```

source node

Definition at line 96 of file [edge.h](#).

Referenced by [Markov::Edge< char >::TraverseNode\(\)](#).

8.8.4.3 _weight

```
template<typename NodeStorageType >
```

```
long int Markov::Edge< NodeStorageType >::_weight [private]
```

target node

Definition at line 97 of file [edge.h](#).

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

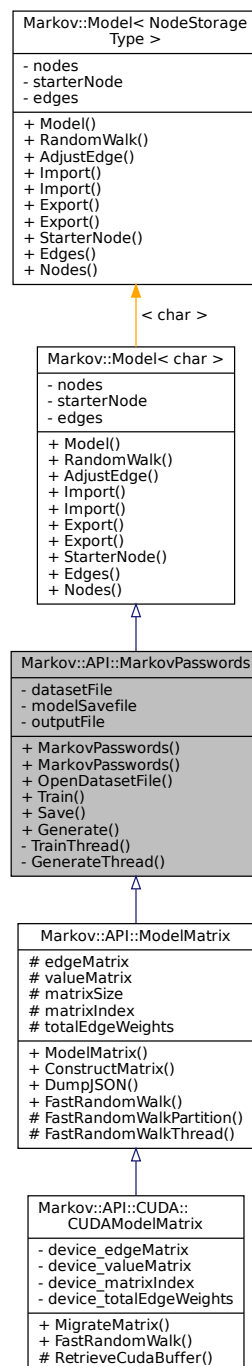
- [model.h](#)
- [edge.h](#)

8.9 Markov::API::MarkovPasswords Class Reference

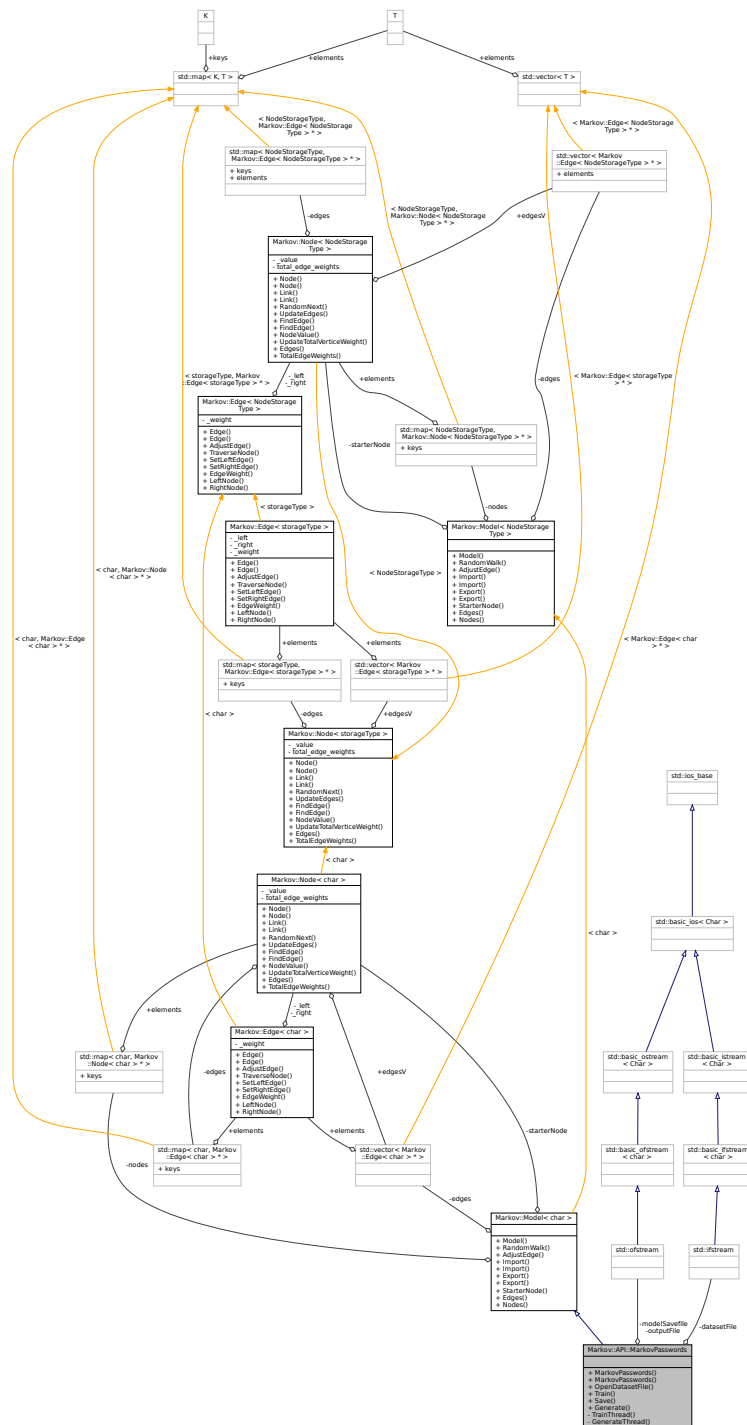
[Markov::Model](#) with char represented nodes.

```
#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.
- 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.

Private Member Functions

- void [TrainThread](#) ([Markov::API::Concurrency::ThreadSharedListHandler](#) *listhandler, char delimiter)
A single thread invoked by the Train function.
- void [GenerateThread](#) (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)
A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * [datasetFile](#)
- std::ofstream * [modelSavefile](#)
- std::ofstream * [outputFile](#)
- 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.

8.9.1 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 17 of file [markovPasswords.h](#).

8.9.2 Constructor & Destructor Documentation

8.9.2.1 MarkovPasswords() [1/2]

Markov::API::MarkovPasswords::MarkovPasswords ()
 Initialize the markov model from MarkovModel::Markov::Model.
 Parent constructor. Has no extra functionality.

Definition at line 10 of file [markovPasswords.cpp](#).

```
00010                                     : Markov::Model<char>() {
00011
00012
00013 }
```

8.9.2.2 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 and empty model, and calling [MarkovPasswords::Import](#) on the filename.

Parameters

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

Example Use: Construction via filename

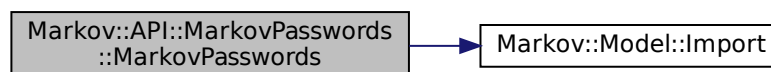
```
MarkovPasswords mp("test.mdl");
```

Definition at line 15 of file [markovPasswords.cpp](#).

```
00015                                     {
00016
00017     std::ifstream* importFile;
00018
00019     this->Import(filename);
00020
00021     //std::ifstream* newFile(filename);
00022
00023     //importFile = newFile;
00024
00025 }
```

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

Here is the call graph for this function:



8.9.3 Member Function Documentation

8.9.3.1 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 322 of file [model.h](#).

```
00322                                     {
00323     NodeStorageType p = payload[0];
00324     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00325     Markov::Edge<NodeStorageType>* e;
00326     int i = 0;
00327
00328     if (p == 0) return;
00329     while (p != 0) {
00330         e = curnode->FindEdge(p);
00331         if (e == NULL) return;
00332         e->AdjustEdge(occurrence);
00333         curnode = e->RightNode();
00334         p = payload[++i];
00335     }
00336
00337     e = curnode->FindEdge('\xff');
00338     e->AdjustEdge(occurrence);
00339     return;
00340 }
```

8.9.3.2 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 172 of file [model.h](#).

```
00172 { return &edges; }
```

8.9.3.3 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 285 of file [model.h](#).

```
00285                                     {
00286     std::ofstream exportfile;
00287     exportfile.open(filename);
00288     return this->Export(&exportfile);
00289 }
```

8.9.3.4 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 273 of file [model.h](#).

```
00273                                     {
00274     Markov::Edge<NodeStorageType>* e;
00275     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00276         e = this->edges[i];
00277         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
00278         e->RightNode()->NodeValue() << "\n";
00279         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
00279         "\n";
00279     }
00280
00281     return true;
00282 }
```

8.9.3.5 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 92 of file [markovPasswords.cpp](#).

```
00092                                     {
00093     char* res;
00094     char print[100];
00095     std::ofstream wordlist;
00096     wordlist.open(wordlistFileName);
00097     std::mutex mlock;
00098     int iterationsPerThread = n/threads;
00099     int iterationsCarryOver = n%threads;
00100     std::vector<std::thread*> threadsV;
00101     for (int i=0;i<threads;i++){
00102         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
&mlock, iterationsPerThread, &wordlist, minLen, maxLen));
```

```

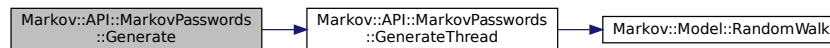
00103     }
00104
00105     for(int i=0;i<threads;i++){
00106         threadsV[i]->join();
00107         delete threadsV[i];
00108     }
00109
00110     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00111
00112 }

```

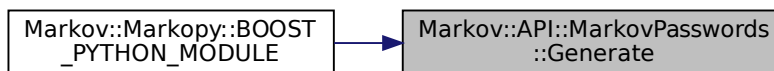
References [GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.3.6 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

<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 114 of file [markovPasswords.cpp](#).

```

00114
00115     {
00116         char* res = new char[maxLen+5];
00117         if(n==0) return;
00118         Markov::Random::Marsaglia MarsagliaRandomEngine;
00119         for (int i = 0; i < n; i++) {
00120             this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);

```

```

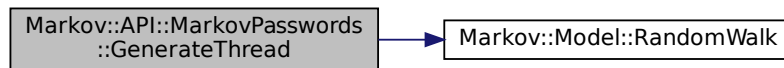
00121         outputLock->lock();
00122         *wordlist « res « "\n";
00123         outputLock->unlock();
00124     }
00125 }

```

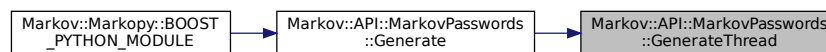
References [Markov::Model< NodeStorageType >::RandomWalk\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.3.7 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 265 of file [model.h](#).

```

00265                                     {
00266         std::ifstream importfile;
00267         importfile.open(filename);
00268         return this->Import(&importfile);
00269     }
00270 }

```

8.9.3.8 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 206 of file [model.h](#).

```
00206                                     {
00207     std::string cell;
00208
00209     char src;
00210     char target;
00211     long int oc;
00212
00213     while (std::getline(*f, cell)) {
00214         //std::cout << "cell: " << cell << std::endl;
00215         src = cell[0];
00216         target = cell[cell.length() - 1];
00217         char* j;
00218         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00219         //std::cout << oc << "\n";
00220         Markov::Node<NodeStorageType>* srcN;
00221         Markov::Node<NodeStorageType>* targetN;
00222         Markov::Edge<NodeStorageType>* e;
00223         if (this->nodes.find(src) == this->nodes.end()) {
00224             srcN = new Markov::Node<NodeStorageType>(src);
00225             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00226             //std::cout << "Creating new node at start.\n";
00227         }
00228         else {
00229             srcN = this->nodes.find(src)->second;
00230         }
00231
00232         if (this->nodes.find(target) == this->nodes.end()) {
00233             targetN = new Markov::Node<NodeStorageType>(target);
00234             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00235             //std::cout << "Creating new node at end.\n";
00236         }
00237         else {
00238             targetN = this->nodes.find(target)->second;
00239         }
00240         e = srcN->Link(targetN);
00241         e->AdjustEdge(oc);
00242         this->edges.push_back(e);
00243
00244         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00245         int(targetN->NodeValue()) << "\n";
00246
00247     }
00248
00249     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00250         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00251         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
00252             Edge<NodeStorageType> *rhs)->bool{
00253             return lhs->EdgeWeight() > rhs->EdgeWeight();
00254         });
00255         //for(int i=0;i<x.second->edgesV.size();i++)
00256         // std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00257         //std::cout << "\n";
00258     }
00259     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00260     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00261     return true;
00262 }
```

8.9.3.9 Nodes()

```
std::map<char , Node<char >*>* Markov::Model< char >::Nodes [inline], [inherited]
```

Return starter [Node](#).

Returns

starter node with 00 NodeValue

Definition at line 177 of file [model.h](#).

```
00177 { return &nodes; }
```

8.9.3.10 OpenDatasetFile()

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename )
```

Open dataset file and return the ifstream pointer.

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

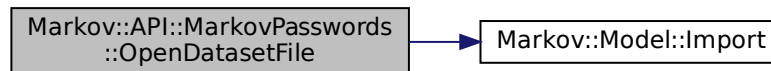
ifstream* to the the dataset file

Definition at line 27 of file [markovPasswords.cpp](#).

```
00027
00028
00029     std::ifstream* datasetFile;
00030
00031     std::ifstream newFile(filename);
00032
00033     datasetFile = &newFile;
00034
00035     this->Import(datasetFile);
00036     return datasetFile;
00037 }
```

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

Here is the call graph for this function:



8.9.3.11 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 criateria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
```

```

    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}

```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 292 of file [model.h](#).

```

00292
00293     Markov::Node<NodeStorageType>* n = this->starterNode;
00294     int len = 0;
00295     Markov::Node<NodeStorageType>* temp_node;
00296     while (true) {
00297         temp_node = n->RandomNext(randomEngine);
00298         if (len >= maxSetting) {
00299             break;
00300         }
00301         else if ((temp_node == NULL) && (len < minSetting)) {
00302             continue;
00303         }
00304         else if (temp_node == NULL) {
00305             break;
00306         }
00307     }
00308     n = temp_node;
00309     buffer[len++] = n->NodeValue();
00310 }
00311
00312 //null terminate the string
00313 buffer[len] = 0x00;
00314
00315 //do something with the generated string
00316 return buffer; //for now
00317 }

```

8.9.3.12 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 80 of file [markovPasswords.cpp](#).

```

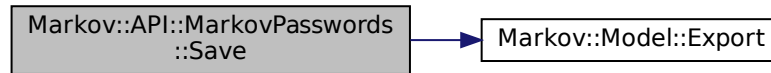
00080
00081     std::ofstream* exportFile;
00082
00083     std::ofstream newFile(filename);
00084
00085     exportFile = &newFile;
00086
00087     this->Export(exportFile);

```

```
00088     return exportFile;
00089 }
```

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

Here is the call graph for this function:



8.9.3.13 StarterNode()

[Node<char >*](#) [Markov::Model< char >::StarterNode](#) [inline], [inherited]

Return starter [Node](#).

Returns

starter node with 00 NodeValue

Definition at line 167 of file [model.h](#).

```
00167 { return starterNode; }
```

8.9.3.14 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");
```

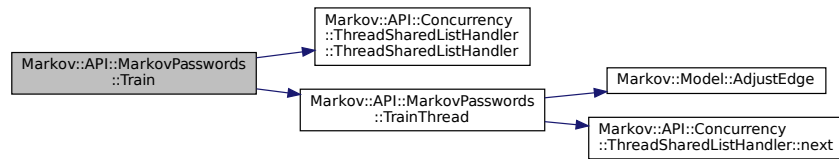
Definition at line 40 of file [markovPasswords.cpp](#).

```
00040
00041     Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
00042     auto start = std::chrono::high_resolution_clock::now();
00043
00044     std::vector<std::thread*> threadsV;
00045     for(int i=0;i<threads;i++){
00046         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
00047             &listhandler, 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 }
```

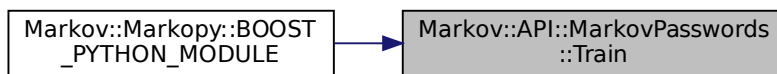
References [Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler\(\)](#), and [TrainThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.3.15 TrainThread()

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

A single thread invoked by the Train function.

Parameters

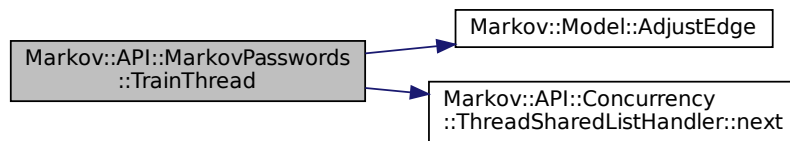
<i>listhandler</i>	- Listhandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

Definition at line 59 of file [markovPasswords.cpp](#).

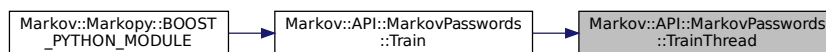
```
00059
{
00060     char format_str[] = "%ld,%s";
00061     format_str[2]=delimiter;
00062     std::string line;
00063     while (listhandler->next(&line)) {
00064         long int oc;
00065         if (line.size() > 100) {
00066             line = line.substr(0, 100);
00067         }
00068         char* linebuf = new char[line.length()+5];
00069 #ifdef _WIN32
00070         sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
00071 #else
00072         sscanf(line.c_str(), format_str, &oc, linebuf);
00073 #endif
00074         this->AdjustEdge((const char*)linebuf, oc);
00075         delete linebuf;
00076     }
00077 }
```

References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), 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:



8.9.4 Member Data Documentation

8.9.4.1 datasetFile

```
std::ifstream* Markov::API::MarkovPasswords::datasetFile [private]
```

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

8.9.4.2 edges

```
std::vector<Edge<char >*> Markov::Model< char >::edges [private], [inherited]
```

A list of all edges in this model.

Definition at line 194 of file [model.h](#).

8.9.4.3 modelSavefile

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

Definition at line 107 of file [markovPasswords.h](#).

8.9.4.4 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 183 of file [model.h](#).

8.9.4.5 outputFile

```
std::ofstream* Markov::API::MarkovPasswords::outputFile [private]
```

Definition at line 108 of file [markovPasswords.h](#).

8.9.4.6 starterNode

`Node<char >*` `Markov::Model< char >::starterNode` [private], [inherited]

Starter `Node` of this model.

Definition at line 188 of file `model.h`.

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

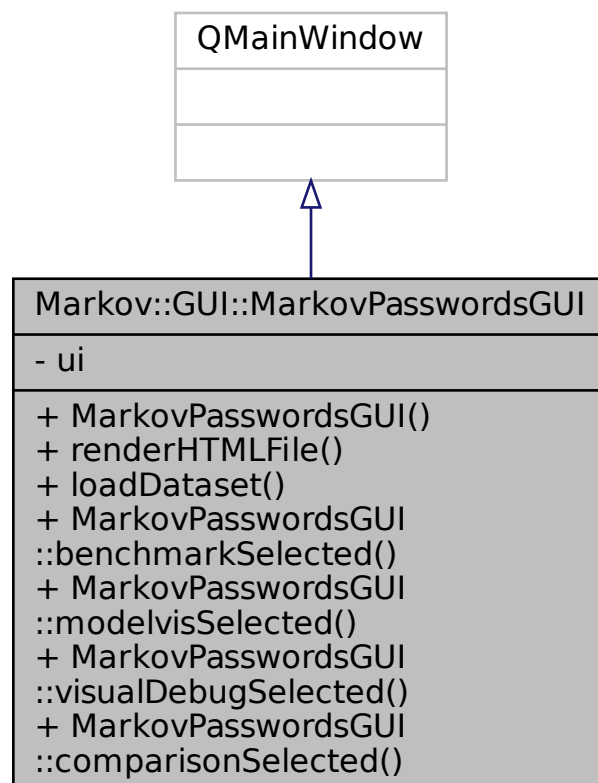
- `markovPasswords.h`
- `markovPasswords.cpp`

8.10 Markov::GUI::MarkovPasswordsGUI Class Reference

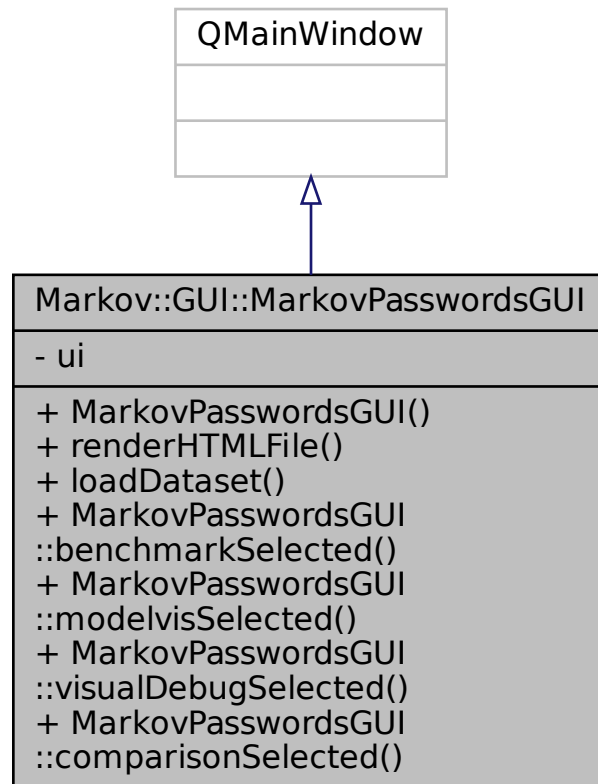
Reporting UI.

```
#include <MarkovPasswordsGUI.h>
```

Inheritance diagram for Markov::GUI::MarkovPasswordsGUI:



Collaboration diagram for Markov::GUI::MarkovPasswordsGUI:



Public Slots

- void [MarkovPasswordsGUI::benchmarkSelected](#) ()
- void [MarkovPasswordsGUI::modelvisSelected](#) ()
- void [MarkovPasswordsGUI::visualDebugSelected](#) ()
- void [MarkovPasswordsGUI::comparisonSelected](#) ()

Public Member Functions

- [MarkovPasswordsGUI](#) (QWidget *parent=Q_NULLPTR)
Default QT constructor.
- void [renderHTMLFile](#) (std::string *filename)
Render a HTML file.
- void [loadDataset](#) (std::string *filename)
Load a dataset to current view..

Private Attributes

- Ui::MarkovPasswordsGUIClass [ui](#)

8.10.1 Detailed Description

Reporting UI.

UI for reporting and debugging tools for MarkovPassword

Definition at line 12 of file [MarkovPasswordsGUI.h](#).

8.10.2 Constructor & Destructor Documentation

8.10.2.1 MarkovPasswordsGUI()

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

Default QT constructor.

Parameters

<i>parent</i>	- Parent widget.
---------------	------------------

Definition at line 8 of file [MarkovPasswordsGUI.cpp](#).

```
00009 : QMainWindow(parent)
00010 {
00011     ui.setupUi(this);
00012
00013
00014     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {benchmarkSelected(); });
00015     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {modelvisSelected(); });
00016     QObject::connect(ui.pushButton_4, &QPushButton::clicked, this, [this] {comparisonSelected(); });
00017 }
```

8.10.3 Member Function Documentation

8.10.3.1 loadDataset()

```
void MarkovPasswordsGUI::loadDataset (
    std::string * filename )
```

Load a dataset to current view..

Parameters

<i>filename</i>	- Filename of the dataset file. (relative path to the views folder).
-----------------	--

Definition at line 78 of file [MarkovPasswordsGUI.cpp](#).

```
00078 {
00079     //extract and parametrize the code from constructor
00080
00081 }
```

8.10.3.2 MarkovPasswordsGUI::benchmarkSelected

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

8.10.3.3 MarkovPasswordsGUI::comparisonSelected

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

8.10.3.4 MarkovPasswordsGUI::modelvisSelected

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

8.10.3.5 MarkovPasswordsGUI::visualDebugSelected

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

8.10.3.6 renderHTMLFile()

```
void MarkovPasswordsGUI::renderHTMLFile (
    std::string * filename )
```

Render a HTML file.

Parameters

<i>filename</i>	- Filename of the html file. (relative path to the views folder).
-----------------	---

Definition at line 71 of file [MarkovPasswordsGUI.cpp](#).

```
00071                                     {
00072     //extract and parametrize the code from constructor
00073
00074 }
```

8.10.4 Member Data Documentation

8.10.4.1 ui

```
Ui::MarkovPasswordsGUIClass Markov::GUI::MarkovPasswordsGUI::ui [private]
```

Definition at line 32 of file [MarkovPasswordsGUI.h](#).

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

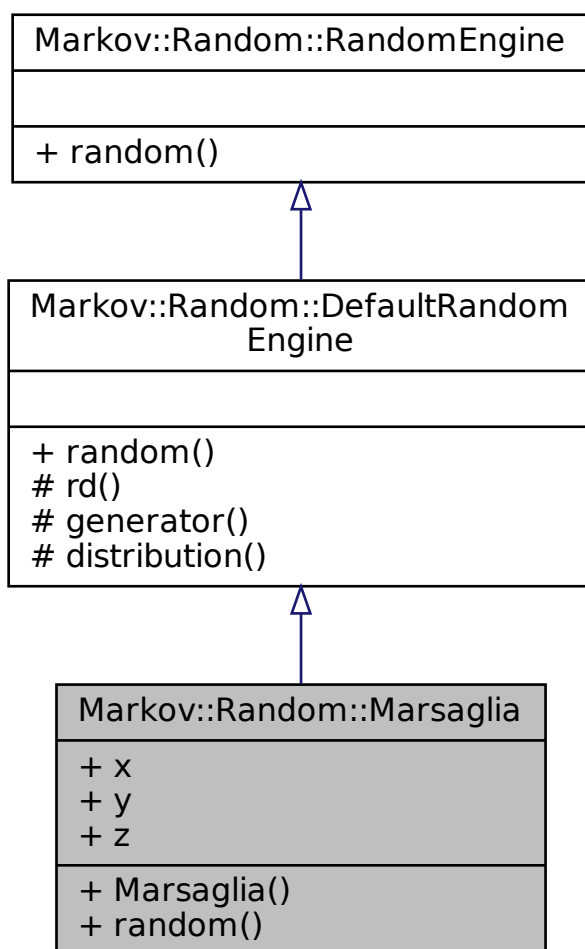
- [MarkovPasswordsGUI.h](#)
- [MarkovPasswordsGUI.cpp](#)

8.11 Markov::Random::Marsaglia Class Reference

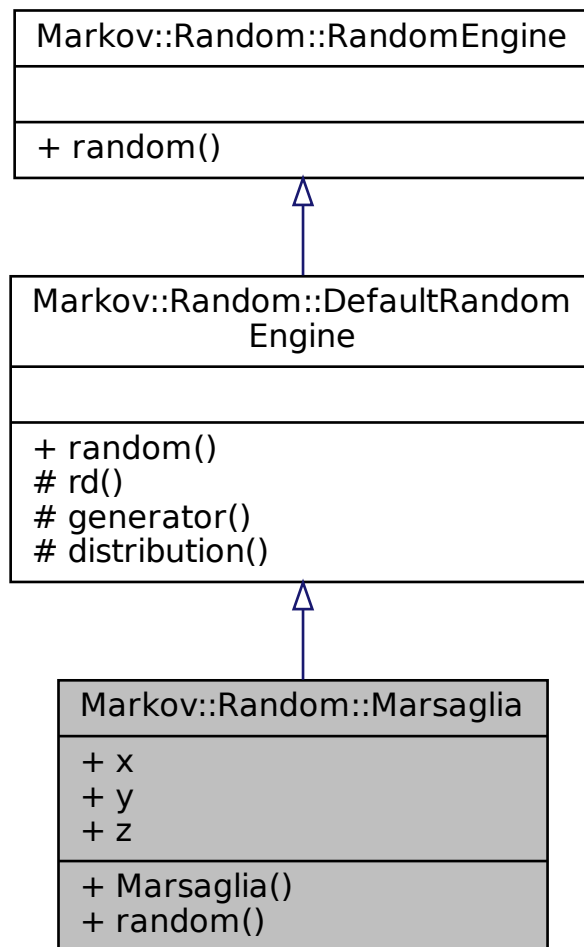
Implementation of [Marsaglia Random](#) Engine.

```
#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.

8.11.1 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 116 of file [random.h](#).

8.11.2 Constructor & Destructor Documentation

8.11.2.1 Marsaglia()

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

Construct [Marsaglia](#) Engine.

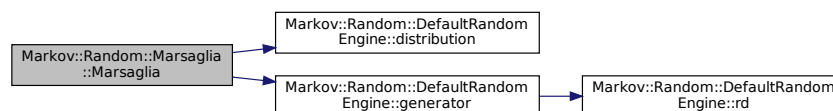
Initialize x,y and z using the default random engine.

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

```
00123     {
00124         this->x = this->distribution() (this->generator());
00125         this->y = this->distribution() (this->generator());
00126         this->z = this->distribution() (this->generator());
00127         //std::cout << "x: " << x << ", y: " << y << ", z: " << z << "\n";
00128     }
```

References [Markov::Random::DefaultRandomEngine::distribution\(\)](#), [Markov::Random::DefaultRandomEngine::generator\(\)](#), [x](#), [y](#), and [z](#).

Here is the call graph for this function:



8.11.3 Member Function Documentation

8.11.3.1 distribution()

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

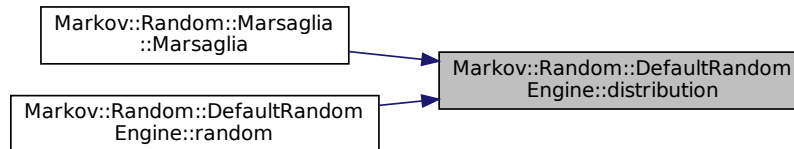
Distribution schema for seeding.

Definition at line 81 of file [random.h](#).

```
00081                                     {
00082         static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffffff);
00083         return _distribution;
00084     }
```

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

Here is the caller graph for this function:



8.11.3.2 generator()

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

Default random engine for seeding.

Definition at line 73 of file [random.h](#).

```
00073                                     {
00074         static std::default_random_engine _generator(rd() ());
00075         return _generator;
00076     }
```

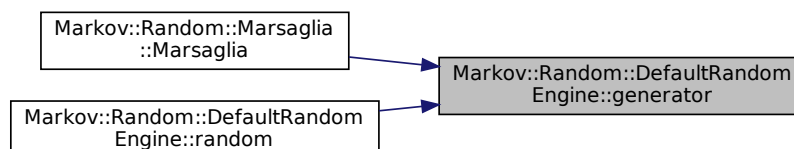
References [Markov::Random::DefaultRandomEngine::rd\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.11.3.3 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 131 of file [random.h](#).

```

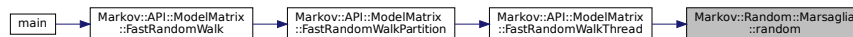
00131         {
00132             unsigned long t;
00133             x ^= x << 16;
00134             x ^= x >> 5;
00135             x ^= x << 1;
00136
00137             t = x;
00138             x = y;
00139             y = z;
00140             z = t ^ x ^ y;
00141
00142             return z;
00143         }

```

References [x](#), [y](#), and [z](#).

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

Here is the caller graph for this function:



8.11.3.4 rd()

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

Default random device for seeding.

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

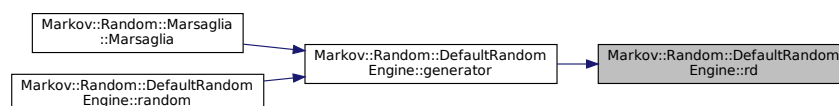
```

00065         {
00066             static std::random_device _rd;
00067             return _rd;
00068         }

```

Referenced by [Markov::Random::DefaultRandomEngine::generator\(\)](#).

Here is the caller graph for this function:



8.11.4 Member Data Documentation

8.11.4.1 x

`unsigned long Markov::Random::Marsaglia::x`

Definition at line 146 of file [random.h](#).

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

8.11.4.2 y

`unsigned long Markov::Random::Marsaglia::y`

Definition at line 147 of file [random.h](#).

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

8.11.4.3 z

```
unsigned long Markov::Random::Marsaglia::z
```

Definition at line 148 of file [random.h](#).

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

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

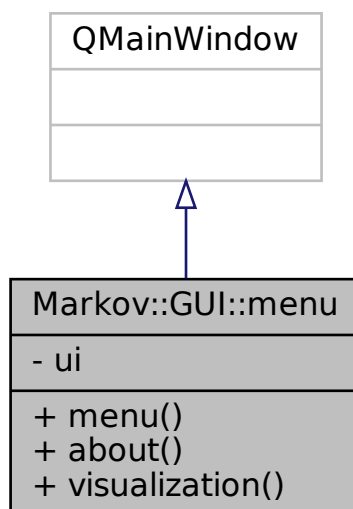
- [random.h](#)

8.12 Markov::GUI::menu Class Reference

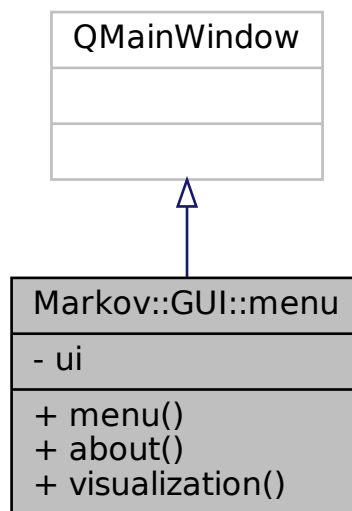
QT Menu class.

```
#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](#)

8.12.1 Detailed Description

QT Menu class.

Definition at line 9 of file [menu.h](#).

8.12.2 Constructor & Destructor Documentation

8.12.2.1 menu()

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

Definition at line 8 of file [menu.cpp](#).

```
00009 : QMainWindow(parent)
00010 {
00011     ui.setupUi(this);
00012
00013
00014     //QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {about(); });
00015     QObject::connect(ui.visu, &QPushButton::clicked, this, [this] {visualization(); });
00016 }
```

8.12.3 Member Function Documentation

8.12.3.1 about

```
void menu::about ( ) [slot]
Definition at line 17 of file menu.cpp.
00017         {
00018
00019
00020 }
```

8.12.3.2 visualization

```
void menu::visualization ( ) [slot]
Definition at line 21 of file menu.cpp.
00021         {
00022     MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
00023     w->show();
00024     this->close();
00025 }
```

8.12.4 Member Data Documentation

8.12.4.1 ui

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

Definition at line 15 of file menu.h.

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

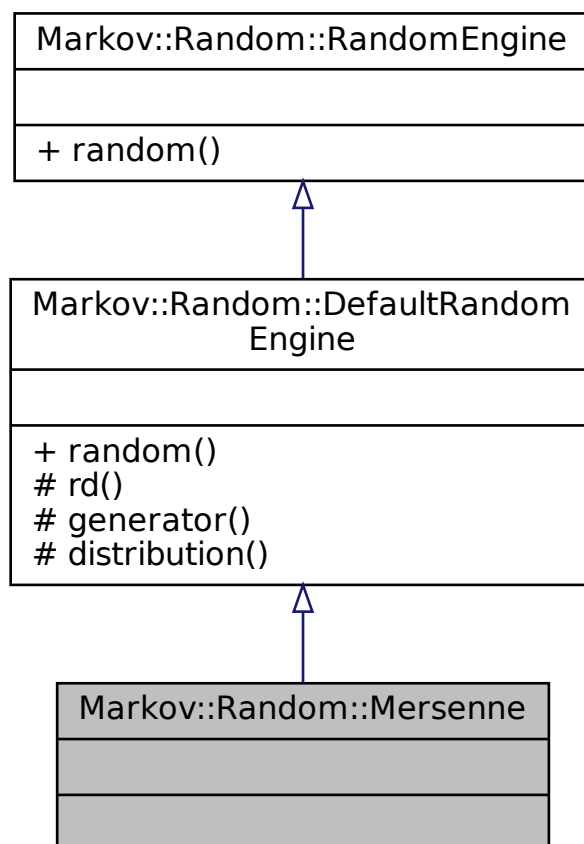
- [menu.h](#)
- [menu.cpp](#)

8.13 Markov::Random::Mersenne Class Reference

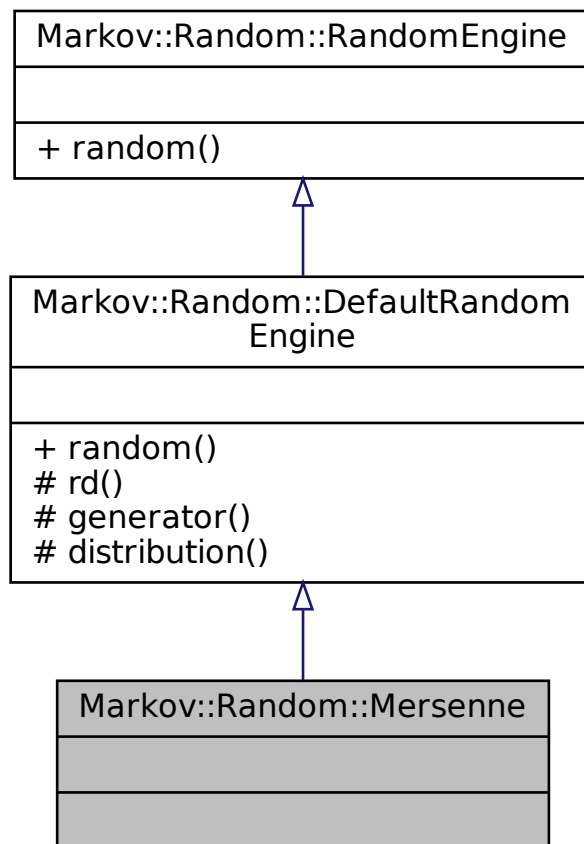
Implementation of [Mersenne](#) Twister Engine.

```
#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.

8.13.1 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 176 of file [random.h](#).

8.13.2 Member Function Documentation

8.13.2.1 distribution()

```

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

```

Distribution schema for seeding.

Definition at line 81 of file [random.h](#).

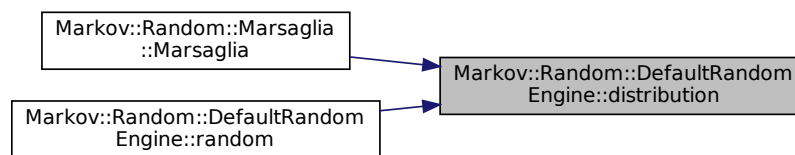
```

00081                                     {
00082         static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffffff);
00083         return _distribution;
00084     }

```

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

Here is the caller graph for this function:



8.13.2.2 generator()

```

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

```

Default random engine for seeding.

Definition at line 73 of file [random.h](#).

```

00073                                     {
00074         static std::default_random_engine _generator(rd() ());
00075         return _generator;
00076     }

```

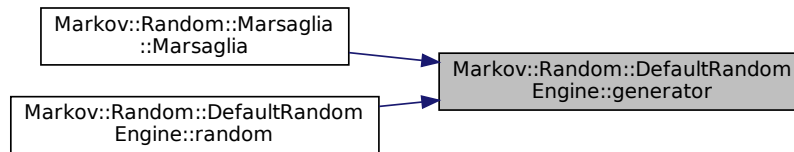
References [Markov::Random::DefaultRandomEngine::rd\(\)](#).

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:



8.13.2.3 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](#).

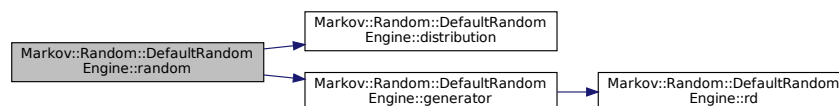
Definition at line 57 of file [random.h](#).

```

00057         {
00058             return this->distribution() (this->generator());
00059         }
  
```

References [Markov::Random::DefaultRandomEngine::distribution\(\)](#), and [Markov::Random::DefaultRandomEngine::generator\(\)](#).

Here is the call graph for this function:



8.13.2.4 rd()

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

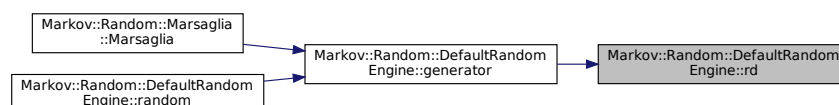
Definition at line 65 of file [random.h](#).

```

00065         {
00066             static std::random_device _rd;
00067             return _rd;
00068         }
  
```

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:

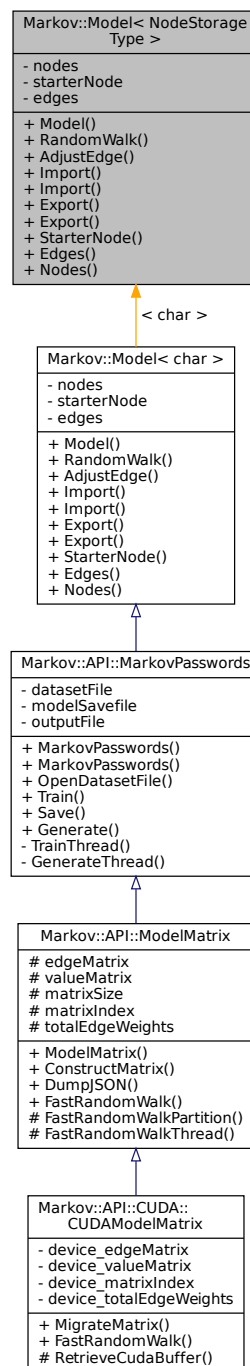
- [random.h](#)

8.14 Markov::Model< NodeStorageType > Class Template Reference

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

```
#include <model.h>
```

Inheritance diagram for Markov::Model< NodeStorageType >:



- *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](#).

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.

8.14.1 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 41 of file [model.h](#).

8.14.2 Constructor & Destructor Documentation

8.14.2.1 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 200 of file [model.h](#).

```
00200         {
00201     this->starterNode = new Markov::Node<NodeStorageType>(0);
00202     this->nodes.insert({ 0, this->starterNode });
00203 }
```

8.14.3 Member Function Documentation

8.14.3.1 AdjustEdge()

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

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

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

Definition at line 322 of file [model.h](#).

```
00322
00323     NodeStorageType p = payload[0];
00324     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00325     Markov::Edge<NodeStorageType>* e;
00326     int i = 0;
00327
00328     if (p == 0) return;
00329     while (p != 0) {
00330         e = curnode->FindEdge(p);
00331         if (e == NULL) return;
00332         e->AdjustEdge(occurrence);
00333         curnode = e->RightNode();
00334         p = payload[++i];
00335     }
00336
00337     e = curnode->FindEdge('\xff');
00338     e->AdjustEdge(occurrence);
00339     return;
00340 }
```

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

Here is the caller graph for this function:



8.14.3.2 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 172 of file [model.h](#).

```
00172 { return &edges; }
```

8.14.3.3 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 285 of file [model.h](#).

```
00285                                     {
00286     std::ofstream exportfile;
00287     exportfile.open(filename);
00288     return this->Export(&exportfile);
00289 }
```

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the caller graph for this function:



8.14.3.4 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

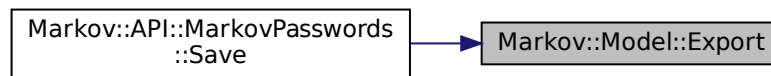
```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 273 of file [model.h](#).

```
00273                                     {
00274     Markov::Edge<NodeStorageType>* e;
00275     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00276         e = this->edges[i];
00277         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
00278         e->RightNode()->NodeValue() << "\n";
00279         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
00279         "\n";
00280     }
00281     return true;
00282 }
```

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

Here is the caller graph for this function:



8.14.3.5 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");
  
```

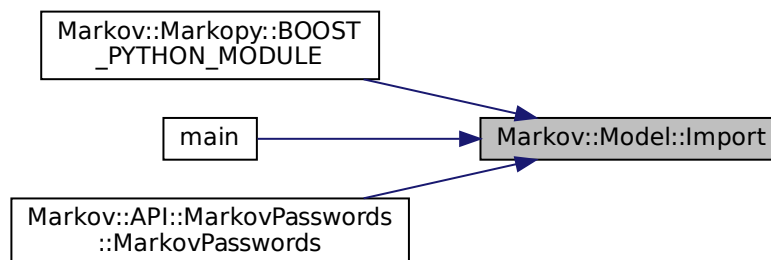
Definition at line 265 of file [model.h](#).

```

00265                                     {
00266     std::ifstream importfile;
00267     importfile.open(filename);
00268     return this->Import(&importfile);
00269 }
00270
  
```

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), [main\(\)](#), and [Markov::API::MarkovPasswords::MarkovPasswords\(\)](#).

Here is the caller graph for this function:



8.14.3.6 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

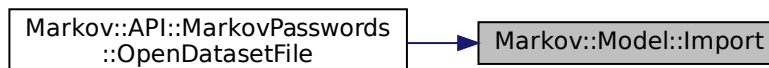
```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 206 of file [model.h](#).

```
00206                                     {
00207     std::string cell;
00208
00209     char src;
00210     char target;
00211     long int oc;
00212
00213     while (std::getline(*f, cell)) {
00214         //std::cout << "cell: " << cell << std::endl;
00215         src = cell[0];
00216         target = cell[cell.length() - 1];
00217         char* j;
00218         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00219         //std::cout << oc << "\n";
00220         Markov::Node<NodeStorageType>* srcN;
00221         Markov::Node<NodeStorageType>* targetN;
00222         Markov::Edge<NodeStorageType>* e;
00223         if (this->nodes.find(src) == this->nodes.end()) {
00224             srcN = new Markov::Node<NodeStorageType>(src);
00225             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00226             //std::cout << "Creating new node at start.\n";
00227         }
00228         else {
00229             srcN = this->nodes.find(src)->second;
00230         }
00231
00232         if (this->nodes.find(target) == this->nodes.end()) {
00233             targetN = new Markov::Node<NodeStorageType>(target);
00234             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00235             //std::cout << "Creating new node at end.\n";
00236         }
00237         else {
00238             targetN = this->nodes.find(target)->second;
00239         }
00240         e = srcN->Link(targetN);
00241         e->AdjustEdge(oc);
00242         this->edges.push_back(e);
00243
00244         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00245         int(targetN->NodeValue()) << "\n";
00246     }
00247
00248     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00249         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00250         std::sort(x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
00251             Edge<NodeStorageType> *rhs)->bool{
00252             return lhs->EdgeWeight() > rhs->EdgeWeight();
00253         });
00254         //for(int i=0; i<x.second->edgesV.size(); i++)
00255         //    std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00256         //std::cout << "\n";
00257     }
00258     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00259     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00260
00261     return true;
00262 }
```

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

Here is the caller graph for this function:



8.14.3.7 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 177 of file [model.h](#).

```
00177 { return &nodes; }
```

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

Here is the caller graph for this function:



8.14.3.8 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 criateria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate

Parameters

<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 292 of file [model.h](#).

```

00292
00293     Markov::Node<NodeStorageType>* n = this->starterNode;
00294     int len = 0;
00295     Markov::Node<NodeStorageType>* temp_node;
00296     while (true) {
00297         temp_node = n->RandomNext (randomEngine);
00298         if (len >= maxSetting) {
00299             break;
00300         }
00301         else if ((temp_node == NULL) && (len < minSetting)) {
00302             continue;
00303         }
00304         else if (temp_node == NULL) {
00305             break;
00306         }
00307     }
00308     n = temp_node;
00309     buffer[len++] = n->NodeValue();
00310 }
00311 //null terminate the string
00312 buffer[len] = 0x00;
00313 //do something with the generated string
00314 return buffer; //for now
00315 }

```

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

Here is the caller graph for this function:



8.14.3.9 StarterNode()

```

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

```

Return starter [Node](#).

Returns

starter node with 00 NodeValue

Definition at line 167 of file [model.h](#).

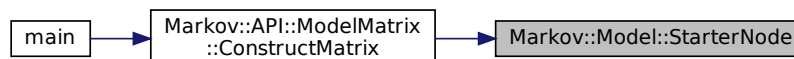
```

00167 { return starterNode; }

```

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

Here is the caller graph for this function:



8.14.4 Member Data Documentation

8.14.4.1 edges

```
template<typename NodeStorageType >
std::vector<Edge<NodeStorageType>*> Markov::Model< NodeStorageType >::edges [private]
```

A list of all edges in this model.

Definition at line 194 of file [model.h](#).

Referenced by [Markov::Model< char >::Edges\(\)](#).

8.14.4.2 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 183 of file [model.h](#).

Referenced by [Markov::Model< char >::Nodes\(\)](#).

8.14.4.3 starterNode

```
template<typename NodeStorageType >
Node<NodeStorageType>* Markov::Model< NodeStorageType >::starterNode [private]
```

Starter Node of this model.

Definition at line 188 of file [model.h](#).

Referenced by [Markov::Model< char >::StarterNode\(\)](#).

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

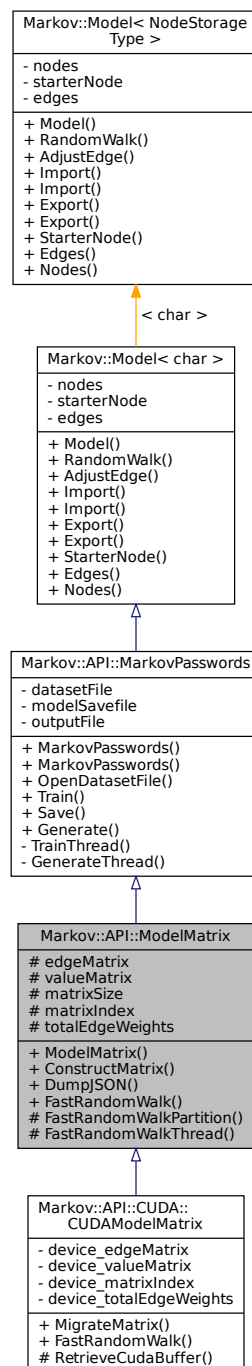
- [model.h](#)

8.15 Markov::API::ModelMatrix Class Reference

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

```
#include <modelMatrix.h>
```


Inheritance diagram for Markov::API::ModelMatrix:



- void [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](#).
- 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.
- 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.

Protected Member Functions

- 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](#).

Protected Attributes

- char ** [edgeMatrix](#)
- long int ** [valueMatrix](#)
- int [matrixSize](#)
- char * [matrixIndex](#)
- long int * [totalEdgeWeights](#)

Private Member Functions

- void [TrainThread](#) ([Markov::API::Concurrency::ThreadSharedListHandler](#) *listhandler, char delimiter)
A single thread invoked by the Train function.
- void [GenerateThread](#) (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)
A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * [datasetFile](#)
- std::ofstream * [modelSavefile](#)
- std::ofstream * [outputFile](#)
- 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.

8.15.1 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 13 of file [modelMatrix.h](#).

8.15.2 Constructor & Destructor Documentation

8.15.2.1 ModelMatrix()

```
Markov::API::ModelMatrix::ModelMatrix ( )
Definition at line 6 of file modelMatrix.cpp.
00006 {
00007
00008 }
```

8.15.3 Member Function Documentation

8.15.3.1 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

<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 322 of file [model.h](#).

```

00322                                     {
00323     NodeStorageType p = payload[0];
00324     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00325     Markov::Edge<NodeStorageType>* e;
00326     int i = 0;
00327
00328     if (p == 0) return;
00329     while (p != 0) {
00330         e = curnode->FindEdge(p);
00331         if (e == NULL) return;
00332         e->AdjustEdge(occurrence);
00333         curnode = e->RightNode();
00334         p = payload[++i];
00335     }
00336
00337     e = curnode->FindEdge('\xff');
00338     e->AdjustEdge(occurrence);
00339     return;
00340 }
```

8.15.3.2 ConstructMatrix()

```
void 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](#).

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

```

00011                                     {
00012     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00013
00014     this->matrixIndex = new char[this->matrixSize];
00015     this->totalEdgeWeights = new long int[this->matrixSize];
00016
00017     this->edgeMatrix = new char*[this->matrixSize];
00018     for(int i=0;i<this->matrixSize;i++){
00019         this->edgeMatrix[i] = new char[this->matrixSize];
00020     }
00021     this->valueMatrix = new long int*[this->matrixSize];
00022     for(int i=0;i<this->matrixSize;i++){
00023         this->valueMatrix[i] = new long int[this->matrixSize];
00024     }
00025     std::map< char, Node< char > * > *nodes;
00026     nodes = this->Nodes();
00027     int i=0;
00028     for (auto const& [repr, node] : *nodes){
00029         if(repr!=0) this->matrixIndex[i] = repr;
00030         else this->matrixIndex[i] = 199;
00031         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00032         for(int j=0;j<this->matrixSize;j++){
00033             char val = node->NodeValue();
00034             if(val < 0){
00035                 for(int k=0;k<this->matrixSize;k++){
00036                     this->valueMatrix[i][k] = 0;
00037                     this->edgeMatrix[i][k] = 255;
00038                 }
00039                 break;
00040             }
00041             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00042                 this->valueMatrix[i][j] = 0;
00043                 this->edgeMatrix[i][j] = 255;
00044             }else if(j==(this->matrixSize-1)) {
00045                 this->valueMatrix[i][j] = 0;
00046                 this->edgeMatrix[i][j] = 255;
00047             }else{
00048                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00049                 this->edgeMatrix[i][j] = node->edgesV[j]->RightNode()->NodeValue();
00050             }
00051         }
00052     }
```

```

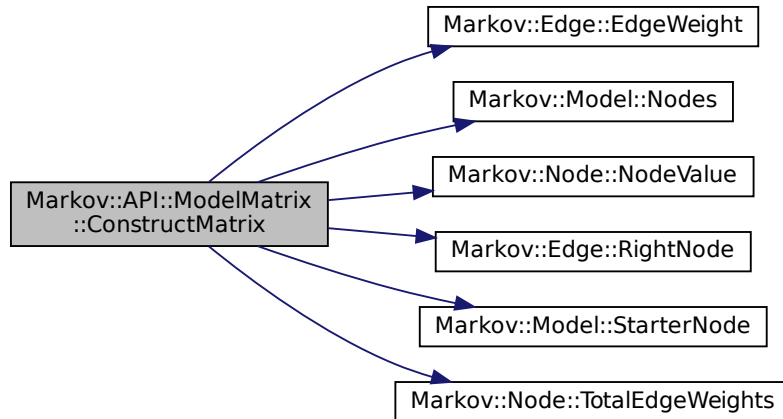
00052         }
00053         i++;
00054     }
00055
00056     //this->DumpJSON();
00057 }

```

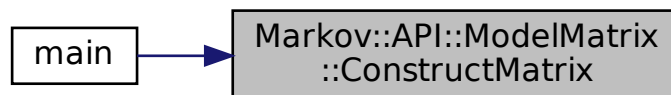
References [edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [matrixIndex](#), [matrixSize](#), [Markov::Model< NodeStorageType >::Nodes](#), [Markov::Node< storageType >::NodeValue\(\)](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Model< NodeStorageType >::totalEdgeWeights](#), [Markov::Node< storageType >::TotalEdgeWeights\(\)](#), and [valueMatrix](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.15.3.3 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 60 of file [modelMatrix.cpp](#).

```

00060     {
00061
00062         std::cout << "{\n  \"index\": \"\";
00063         for(int i=0;i<this->matrixSize;i++){
00064             if(this->matrixIndex[i]=='\"') std::cout << "\\\"";
00065             else if(this->matrixIndex[i]=='\\') std::cout << "\\\"";
00066             else if(this->matrixIndex[i]==0) std::cout << "\\x00";
00067             else if(i==0) std::cout << "\\xff";
00068             else if(this->matrixIndex[i]=='\n') std::cout << "\\n";
00069             else std::cout << this->matrixIndex[i];

```

```

00070     }
00071     std::cout <<
00072     "\",\n"
00073     "    \"edgemap\": {\n";
00074
00075     for(int i=0;i<this->matrixSize;i++){
00076         if(this->matrixIndex[i]=='') std::cout << "        \"\\\"\": [";
00077         else if(this->matrixIndex[i]=='\\') std::cout << "        \"\\\\\"\": [";
00078         else if(this->matrixIndex[i]==0) std::cout << "        \"\\\\\\x00\"\": [";
00079         else if(this->matrixIndex[i]<0) std::cout << "        \"\\\\\\xff\"\": [";
00080         else std::cout << "        \"\" < this->matrixIndex[i] < \"\"\": [";
00081         for(int j=0;j<this->matrixSize;j++){
00082             if(this->edgeMatrix[i][j]=='') std::cout << "            \"\\\"\": ";
00083             else if(this->edgeMatrix[i][j]=='\\') std::cout << "            \"\\\\\"\": ";
00084             else if(this->edgeMatrix[i][j]==0) std::cout << "            \"\\\\\\x00\"\": ";
00085             else if(this->edgeMatrix[i][j]<0) std::cout << "            \"\\\\\\xff\"\": ";
00086             else if(this->matrixIndex[i]=='\\n') std::cout << "            \"\\n\"\": ";
00087             else std::cout << "            \"\" < this->edgeMatrix[i][j] < \"\"\": ";
00088             if(j!=this->matrixSize-1) std::cout << ", ";
00089         }
00090         std::cout << "],\n";
00091     }
00092     std::cout << "},\n";
00093
00094     std::cout << "\"    weightmap\": {\n";
00095     for(int i=0;i<this->matrixSize;i++){
00096         if(this->matrixIndex[i]=='') std::cout << "        \"\\\"\": [";
00097         else if(this->matrixIndex[i]=='\\') std::cout << "        \"\\\\\"\": [";
00098         else if(this->matrixIndex[i]==0) std::cout << "        \"\\\\\\x00\"\": [";
00099         else if(this->matrixIndex[i]<0) std::cout << "        \"\\\\\\xff\"\": [";
00100         else std::cout << "        \"\" < this->matrixIndex[i] < \"\"\": [";
00101
00102         for(int j=0;j<this->matrixSize;j++){
00103             std::cout << this->valueMatrix[i][j];
00104             if(j!=this->matrixSize-1) std::cout << ", ";
00105         }
00106         std::cout << "],\n";
00107     }
00108     std::cout << " }\\n}\\n";
00109 }

```

References [edgeMatrix](#), [matrixIndex](#), [matrixSize](#), and [valueMatrix](#).

8.15.3.4 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 172 of file [model.h](#).

```
00172 { return &edges; }
```

8.15.3.5 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 285 of file [model.h](#).

```

00285
00286     std::ofstream exportfile;
00287     exportfile.open(filename);
00288     return this->Export(&exportfile);
00289 }

```

8.15.3.6 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 273 of file [model.h](#).

```
00273                                     {
00274     Markov::Edge<NodeStorageType>* e;
00275     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00276         e = this->edges[i];
00277         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
00278         e->RightNode()->NodeValue() << "\n";
00279         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
00279         "\n";
00279     }
00280
00281     return true;
00282 }
```

8.15.3.7 FastRandomWalk()

```
void 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 163 of file [modelMatrix.cpp](#).

```
00163
{
00164
```



```

00165
00166     std::ofstream wordlist;
00167     if(bFileIO)
00168         wordlist.open(wordlistFileName);
00169
00170     std::mutex mlock;
00171     if(n<=50000000u) return this->FastRandomWalkPartition(&mlock, &wordlist, n, minLen, maxLen,
00172         bFileIO, threads);
00173     else{
00174         int numberOfPartitions = n/50000000u;
00175         for(int i=0;i<numberOfPartitions;i++)
00176             this->FastRandomWalkPartition(&mlock, &wordlist, 50000000u, minLen, maxLen, bFileIO,
00177         threads);
00178     }
00179 }

```

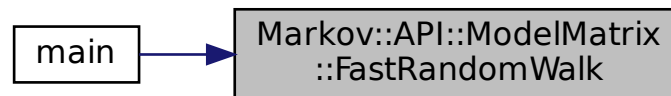
References [FastRandomWalkPartition\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.15.3.8 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 182 of file [modelMatrix.cpp](#).

```

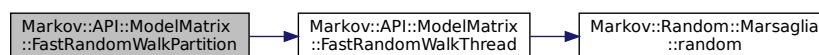
00182
00183                                     {
00184         int iterationsPerThread = n/threads;
00185         int iterationsPerThreadCarryOver = n%threads;
00186
00187         std::vector<std::thread*> threadsV;
00188
00189         int id = 0;
00190         for(int i=0;i<threads;i++){
00191             threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00192 mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00193             id++;
00194         }
00195         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00196 wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00197         for(int i=0;i<threads;i++){
00198             threadsV[i]->join();
00199         }
00200     }

```

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.15.3.9 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>	- DEPRECATED Thread id - No longer used
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 112 of file [modelMatrix.cpp](#).

```

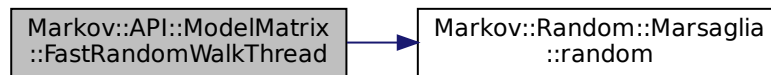
00112
00113         if(n==0) return;
00114
00115         Markov::Random::Marsaglia MarsagliaRandomEngine;
00116         char* e;
00117         char *res = new char[maxLen*n];
00118         int index = 0;
00119         char next;
00120         int len=0;
00121         long int selection;
00122         char cur;
00123         long int bufferctr = 0;
00124         for (int i = 0; i < n; i++) {
00125             cur=199;
00126             len=0;
00127             while (true) {
00128                 e = strchr(this->matrixIndex, cur);
00129                 index = e - this->matrixIndex;
00130                 selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00131                 for(int j=0;j<this->matrixSize;j++){
00132                     selection -= this->valueMatrix[index][j];
00133                     if (selection < 0){
00134                         next = this->edgeMatrix[index][j];
00135                         break;
00136                     }
00137                 }
00138
00139                 if (len >= maxLen) break;
00140                 else if ((next < 0) && (len < minLen)) continue;
00141                 else if (next < 0) break;
00142                 cur = next;
00143                 res[bufferctr + len++] = cur;
00144             }
00145             res[bufferctr + len++] = '\n';
00146             bufferctr+=len;
00147         }
00148
00149         if(bFileIO){
00150             mlock->lock();
00151             *wordlist « res;
00152             mlock->unlock();
00153         }else{
00154             mlock->lock();
00155             std::cout « res;
00156             mlock->unlock();
00157         }
00158         delete res;
00159
00160     }

```

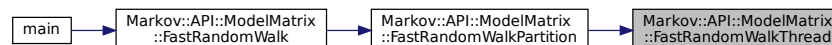
References [edgeMatrix](#), [matrixIndex](#), [matrixSize](#), [Markov::Random::Marsaglia::random\(\)](#), [totalEdgeWeights](#), and [valueMatrix](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.15.3.10 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 92 of file [markovPasswords.cpp](#).

```

00092
{
00093     char* res;
00094     char print[100];
00095     std::ofstream wordlist;
00096     wordlist.open(wordlistFileName);
00097     std::mutex mlock;
00098     int iterationsPerThread = n/threads;
00099     int iterationsCarryOver = n%threads;
00100     std::vector<std::thread*> threadsV;
00101     for(int i=0;i<threads;i++){
00102         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
&mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00103     }
00104
00105     for(int i=0;i<threads;i++){
00106         threadsV[i]->join();
00107         delete threadsV[i];
  
```

```

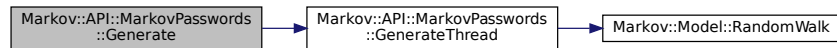
00108     }
00109
00110     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00111
00112 }

```

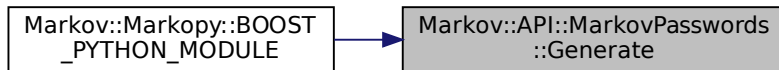
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.15.3.11 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 114 of file [markovPasswords.cpp](#).

```

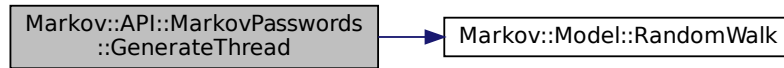
00114
00115     {
00116     char* res = new char[maxLen+5];
00117     if (n==0) return;
00118
00119     Markov::Random::Marsaglia MarsagliaRandomEngine;
00120     for (int i = 0; i < n; i++) {
00121         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00122         outputLock->lock();
00123         *wordlist << res << "\n";
00124         outputLock->unlock();
00125     }
00126 }

```

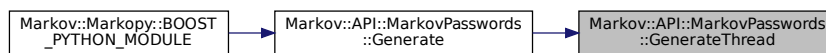
References [Markov::Model< NodeStorageType >::RandomWalk\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.15.3.12 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 265 of file [model.h](#).

```
00265                                     {
00266     std::ifstream importfile;
00267     importfile.open(filename);
00268     return this->Import(&importfile);
00269 }
00270 }
```

8.15.3.13 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 206 of file [model.h](#).

```
00206                                     {
00207     std::string cell;
```

```

00208
00209     char src;
00210     char target;
00211     long int oc;
00212
00213     while (std::getline(*f, cell)) {
00214         //std::cout << "cell: " << cell << std::endl;
00215         src = cell[0];
00216         target = cell[cell.length() - 1];
00217         char* j;
00218         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00219         //std::cout << oc << "\n";
00220         Markov::Node<NodeStorageType>* srcN;
00221         Markov::Node<NodeStorageType>* targetN;
00222         Markov::Edge<NodeStorageType>* e;
00223         if (this->nodes.find(src) == this->nodes.end()) {
00224             srcN = new Markov::Node<NodeStorageType>(src);
00225             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00226             //std::cout << "Creating new node at start.\n";
00227         }
00228         else {
00229             srcN = this->nodes.find(src)->second;
00230         }
00231
00232         if (this->nodes.find(target) == this->nodes.end()) {
00233             targetN = new Markov::Node<NodeStorageType>(target);
00234             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00235             //std::cout << "Creating new node at end.\n";
00236         }
00237         else {
00238             targetN = this->nodes.find(target)->second;
00239         }
00240         e = srcN->Link(targetN);
00241         e->AdjustEdge(oc);
00242         this->edges.push_back(e);
00243
00244         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
int(targetN->NodeValue()) << "\n";
00245
00246     }
00247 }
00248
00249 for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00250     //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00251     std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [] (Edge<NodeStorageType> *lhs,
Edge<NodeStorageType> *rhs)->bool{
00252         return lhs->EdgeWeight() > rhs->EdgeWeight();
00253     });
00254     //for (int i=0; i<x.second->edgesV.size(); i++)
00255     // std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00256     //std::cout << "\n";
00257 }
00258 //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00259 //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00260
00261     return true;
00262 }

```

8.15.3.14 Nodes()

std::map<char , Node<char >*>* Markov::Model< char >::Nodes () [inline], [inherited]

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 177 of file [model.h](#).

```
00177 { return &nodes; }
```

8.15.3.15 OpenDatasetFile()

std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
 const char * filename) [inherited]

Open dataset file and return the ifstream pointer.

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

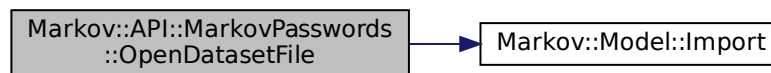
ifstream* to the the dataset file

Definition at line 27 of file [markovPasswords.cpp](#).

```
00027                                     {
00028
00029         std::ifstream* datasetFile;
00030
00031         std::ifstream newFile(filename);
00032
00033         datasetFile = &newFile;
00034
00035         this->Import(datasetFile);
00036         return datasetFile;
00037 }
```

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

Here is the call graph for this function:



8.15.3.16 RandomWalk()

```
char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criateria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```


Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 292 of file [model.h](#).

```

00292
00293     {
00294         Markov::Node<NodeStorageType>* n = this->starterNode;
00295         int len = 0;
00296         Markov::Node<NodeStorageType>* temp_node;
00297         while (true) {
00298             temp_node = n->RandomNext(randomEngine);
00299             if (len >= maxSetting) {
00300                 break;
00301             }
00302             else if ((temp_node == NULL) && (len < minSetting)) {
00303                 continue;
00304             }
00305             else if (temp_node == NULL) {
00306                 break;
00307             }
00308             n = temp_node;
00309             buffer[len++] = n->NodeValue();
00310         }
00311         //null terminate the string
00312         buffer[len] = 0x00;
00313         //do something with the generated string
00314         return buffer; //for now
00315     }
00316
00317
00318
00319 }
```

8.15.3.17 Save()

```

std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
```

Export model to file.

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

std::ofstream* of the exported file.

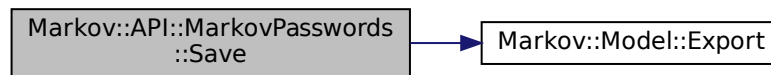
Definition at line 80 of file [markovPasswords.cpp](#).

```

00080
00081     std::ofstream* exportFile;
00082
00083     std::ofstream newFile(filename);
00084
00085     exportFile = &newFile;
00086
00087     this->Export(exportFile);
00088     return exportFile;
00089 }
```

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

Here is the call graph for this function:



8.15.3.18 StarterNode()

`Node<char> * Markov::Model< char >::StarterNode () [inline], [inherited]`

Return starter [Node](#).

Returns

starter node with 00 NodeValue

Definition at line 167 of file [model.h](#).

```
00167 { return starterNode; }
```

8.15.3.19 Train()

```
void Markov::API::MarkovPasswords::Train (
    const char * datasetFileName,
    char delimiter,
    int threads ) [inherited]
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import ("models/2gram.mdl");
mp.Train("password.corpus");
```

Definition at line 40 of file [markovPasswords.cpp](#).

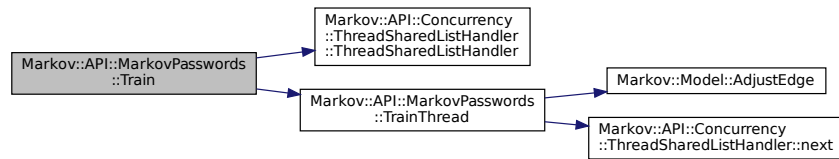
```

00040
00041     Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
00042     auto start = std::chrono::high_resolution_clock::now();
00043
00044     std::vector<std::thread*> threadsV;
00045     for(int i=0;i<threads;i++){
00046         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
00047             &listhandler, 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 }
```

References [Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler\(\)](#), and [Markov::API::MarkovPasswords::](#)

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.15.3.20 TrainThread()

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

A single thread invoked by the Train function.

Parameters

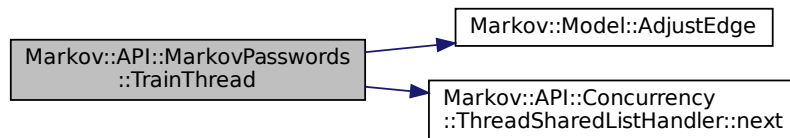
<i>listhandler</i>	- Lishandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

Definition at line 59 of file [markovPasswords.cpp](#).

```
00059
{
00060     char format_str[] = "%ld,%s";
00061     format_str[2]=delimiter;
00062     std::string line;
00063     while (listhandler->next(&line)) {
00064         long int oc;
00065         if (line.size() > 100) {
00066             line = line.substr(0, 100);
00067         }
00068         char* linebuf = new char[line.length()+5];
00069 #ifdef _WIN32
00070         sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
00071 #else
00072         sscanf(line.c_str(), format_str, &oc, linebuf);
00073 #endif
00074         this->AdjustEdge((const char*)linebuf, oc);
00075         delete linebuf;
00076     }
00077 }
```

References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), 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:



8.15.4 Member Data Documentation

8.15.4.1 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile` [private], [inherited]
 Definition at line 106 of file [markovPasswords.h](#).

8.15.4.2 edgeMatrix

`char** Markov::API::ModelMatrix::edgeMatrix` [protected]
 Definition at line 112 of file [modelMatrix.h](#).
 Referenced by [ConstructMatrix\(\)](#), [DumpJSON\(\)](#), and [FastRandomWalkThread\(\)](#).

8.15.4.3 edges

`std::vector<Edge<char >> Markov::Model< char >::edges` [private], [inherited]
 A list of all edges in this model.
 Definition at line 194 of file [model.h](#).

8.15.4.4 matrixIndex

`char* Markov::API::ModelMatrix::matrixIndex` [protected]
 Definition at line 115 of file [modelMatrix.h](#).
 Referenced by [ConstructMatrix\(\)](#), [DumpJSON\(\)](#), and [FastRandomWalkThread\(\)](#).

8.15.4.5 matrixSize

`int Markov::API::ModelMatrix::matrixSize` [protected]
 Definition at line 114 of file [modelMatrix.h](#).
 Referenced by [ConstructMatrix\(\)](#), [DumpJSON\(\)](#), and [FastRandomWalkThread\(\)](#).

8.15.4.6 modelSavefile

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

Definition at line 107 of file [markovPasswords.h](#).

8.15.4.7 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 183 of file [model.h](#).

8.15.4.8 outputFile

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

Definition at line 108 of file [markovPasswords.h](#).

8.15.4.9 starterNode

```
Node<char >* Markov::Model< char >::starterNode [private], [inherited]
```

Starter Node of this model.

Definition at line 188 of file [model.h](#).

8.15.4.10 totalEdgeWeights

```
long int* Markov::API::ModelMatrix::totalEdgeWeights [protected]
```

Definition at line 116 of file [modelMatrix.h](#).

Referenced by [ConstructMatrix\(\)](#), and [FastRandomWalkThread\(\)](#).

8.15.4.11 valueMatrix

```
long int** Markov::API::ModelMatrix::valueMatrix [protected]
```

Definition at line 113 of file [modelMatrix.h](#).

Referenced by [ConstructMatrix\(\)](#), [DumpJSON\(\)](#), and [FastRandomWalkThread\(\)](#).

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

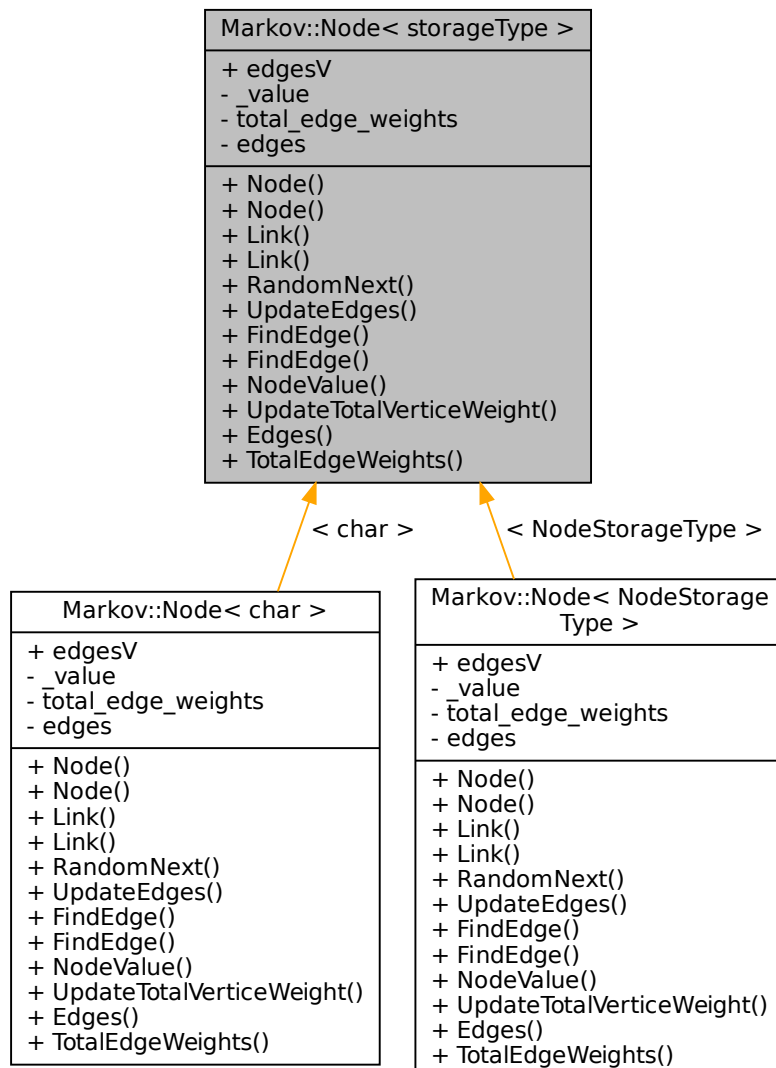
- [modelMatrix.h](#)
- [modelMatrix.cpp](#)

8.16 Markov::Node< storageType > Class Template Reference

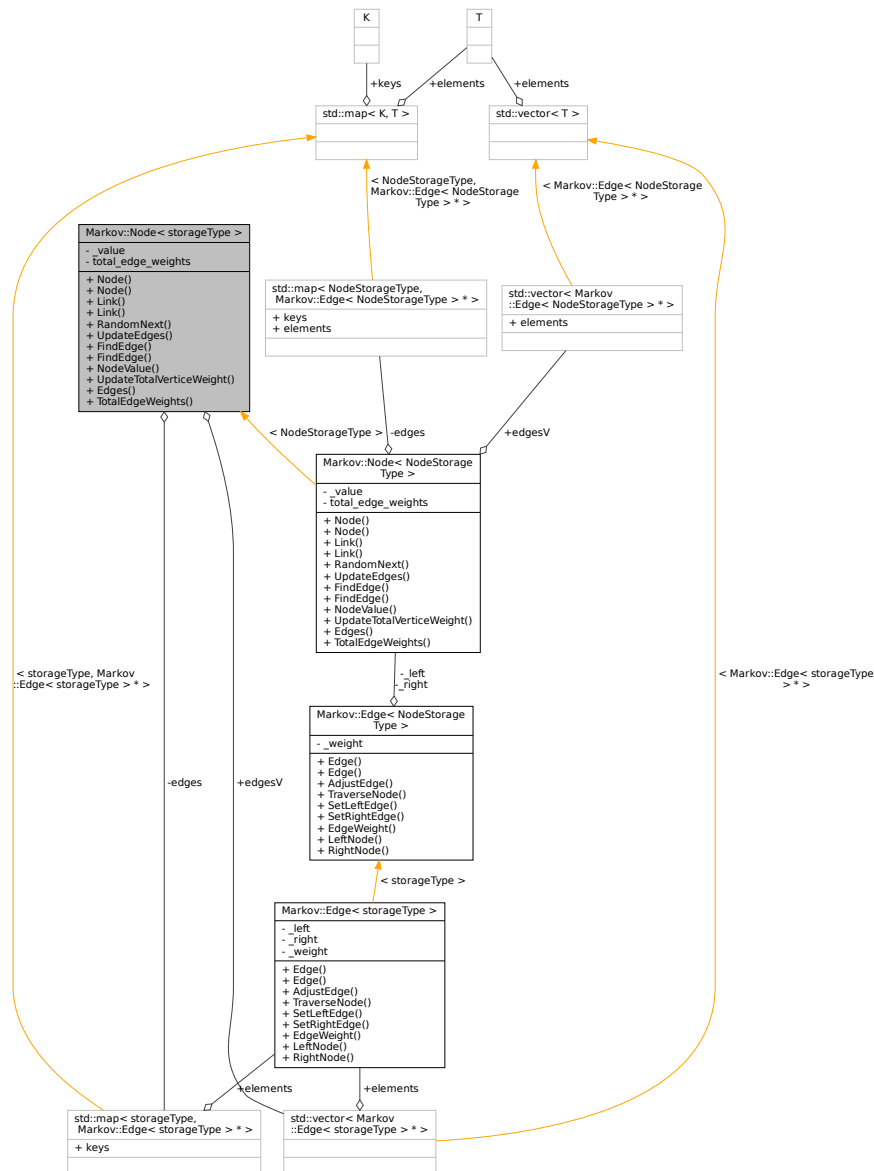
A node class that for the vertices of model. Connected with eachother using [Edge](#).

```
#include <model.h>
```

Inheritance diagram for Markov::Node< storageType >:



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)`
Chose 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 [UpdateTotalVerticeWeight](#) (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](#)
- long int [total_edge_weights](#)
Character representation of this node. 0 for starter, 0xff for terminator.
- std::map< storageType, [Edge](#)< storageType > * > [edges](#)
Total weights of the vertices, required by RandomNext;.

8.16.1 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 23 of file [model.h](#).

8.16.2 Constructor & Destructor Documentation

8.16.2.1 Node() [1/2]

```
template<typename storageType >
Markov::Node< storageType >::Node
Default constructor. Creates an empty Node.
Definition at line 196 of file node.h.
00196     {
00197     this->_value = 0;
00198     this->total_edge_weights = 0L;
00199 };
```

8.16.2.2 Node() [2/2]

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


Parameters

<code>_value</code>	- 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 190 of file [node.h](#).

```
00190                                     {
00191     this->_value = _value;
00192     this->total_edge_weights = 0L;
00193 };
```

8.16.3 Member Function Documentation

8.16.3.1 Edges()

```
template<typename storageType >
std::map< storageType, Markov::Edge< storageType > * > * Markov::Node< storageType >::Edges
[inline]
return edges
```

Definition at line 259 of file [node.h](#).

```
00259                                     {
00260     return &(this->edges);
00261 }
```

8.16.3.2 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.

8.16.3.3 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 247 of file [node.h](#).

```
00247
00248     auto e = this->edges.find(repr);
00249     if (e == this->edges.end()) return NULL;
00250     return e->second;
00251 };
```

8.16.3.4 Link() [1/2]

```
template<typename storageType >
Markov::Edge< storageType > * Markov::Node< storageType >::Link (
    Markov::Edge< storageType > * v )
```

Link this node with another, with this node as its source.

DOES NOT create a new [Edge](#).

Parameters

Edge	- Edge that will accept this node as its LeftNode.
----------------------	--

Returns

the same edge as parameter target.

Example Use: Construct and link nodes

```
Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
LeftNode->Link(e);
```

Definition at line 214 of file [node.h](#).

```
00214
00215     v->SetLeftEdge(this);
00216     this->UpdateEdges(v);
00217     return v;
00218 }
```

8.16.3.5 Link() [2/2]

```
template<typename storageType >
Markov::Edge< storageType > * Markov::Node< storageType >::Link (
    Markov::Node< storageType > * n )
```

Link this node with another, with this node as its source.

Creates a new [Edge](#).

Parameters

<i>target</i>	- Target node which will be the RightNode() of new edge.
---------------	--

Returns

A new node with LeftNode as this, and RightNode as parameter target.

Example Use: Construct nodes

```
Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
```

```
Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
```

Definition at line 207 of file [node.h](#).

```
00207
00208     Markov::Edge<storageType>* v = new Markov::Edge<storageType>(this, n);
00209     this->UpdateEdges(v);
00210     return v;
00211 }
```

8.16.3.6 NodeValue()

```
template<typename storageType >
```

```
unsigned char Markov::Node< storageType >::NodeValue [inline]
```

Return character representation of this node.

Returns

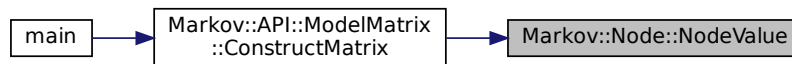
character representation at `_value`.

Definition at line 202 of file [node.h](#).

```
00202
00203     return _value;
00204 }
```

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

Here is the caller graph for this function:



8.16.3.7 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 221 of file `node.h`.

```

00221
00222
00223     //get a random NodeValue in range of total_vertice_weight
00224     long int selection = randomEngine->random() %
this->total_edge_weights; //distribution() (generator()); // distribution(generator);
00225     //make absolute, no negative modulus values wanted
00226     //selection = (selection >= 0) ? selection : (selection + this->total_edge_weights);
00227     for(int i=0; i<this->edgesV.size(); i++){
00228         selection -= this->edgesV[i]->EdgeWeight();
00229         if (selection < 0) return this->edgesV[i]->TraverseNode();
00230     }
00231
00232     //if this assertion is reached, it means there is an implementation error above
00233     std::cout << "This should never be reached (node failed to walk to next)\n"; //cant assert from
child thread
00234     assert(true && "This should never be reached (node failed to walk to next)");
00235     return NULL;
00236 }

```

8.16.3.8 TotalEdgeWeights()

```

template<typename storageType >
long int Markov::Node< storageType >::TotalEdgeWeights [inline]
return total edge weights

```

Definition at line 264 of file `node.h`.

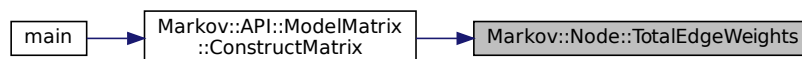
```

00264
00265     return this->total_edge_weights;
00266 }

```

Referenced by `Markov::API::ModelMatrix::ConstructMatrix()`.

Here is the caller graph for this function:



8.16.3.9 UpdateEdges()

```

template<typename storageType >
bool Markov::Node< storageType >::UpdateEdges (
    Markov::Edge< storageType > * v )

```

Insert a new edge to the `this.edges`.

Parameters

<i>edge</i>	- 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 239 of file [node.h](#).

```
00239
00240     this->edges.insert({ v->RightNode()->NodeValue(), v });
00241     this->edgesV.push_back(v);
00242     //this->total_edge_weights += v->EdgeWeight();
00243     return v->TraverseNode();
00244 }
```

8.16.3.10 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 254 of file [node.h](#).

```
00254
00255     this->total_edge_weights += offset;
00256 }
```

8.16.4 Member Data Documentation

8.16.4.1 _value

```
template<typename storageType >
storageType Markov::Node< storageType >::_value [private]
```

Definition at line 169 of file [node.h](#).

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

8.16.4.2 edges

```
template<typename storageType >
std::map<storageType, Edge<storageType>*> Markov::Node< storageType >::edges [private]
```

Total weights of the vertices, required by RandomNext;

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 177 of file [node.h](#).

8.16.4.3 edgesV

```
template<typename storageType >
std::vector<Edge<storageType>*> Markov::Node< storageType >::edgesV
```

Definition at line 165 of file [node.h](#).

8.16.4.4 total_edge_weights

```
template<typename storageType >
long int Markov::Node< storageType >::total_edge_weights [private]
```

Character representation of this node. 0 for starter, 0xff for terminator.

Definition at line 171 of file [node.h](#).

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

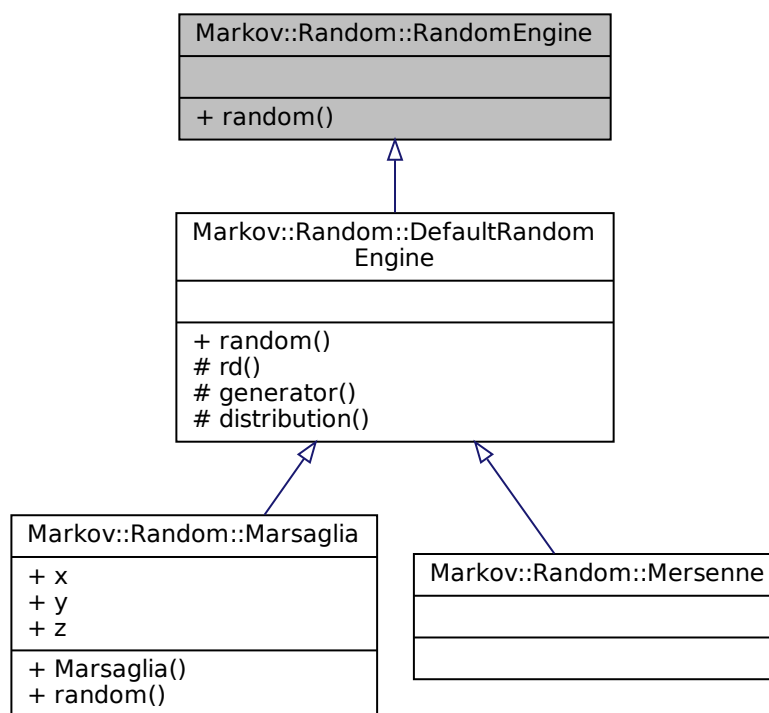
- [model.h](#)
- [node.h](#)

8.17 Markov::Random::RandomEngine Class Reference

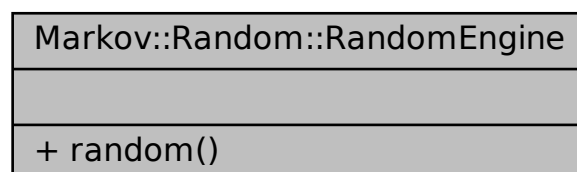
An abstract class for [Random](#) Engine.

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

Inheritance diagram for Markov::Random::RandomEngine:



Collaboration diagram for Markov::Random::RandomEngine:



Public Member Functions

- virtual unsigned long [random](#) ()=0

8.17.1 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 21 of file [random.h](#).

8.17.2 Member Function Documentation

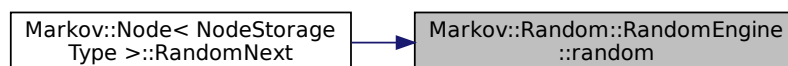
8.17.2.1 random()

```
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:

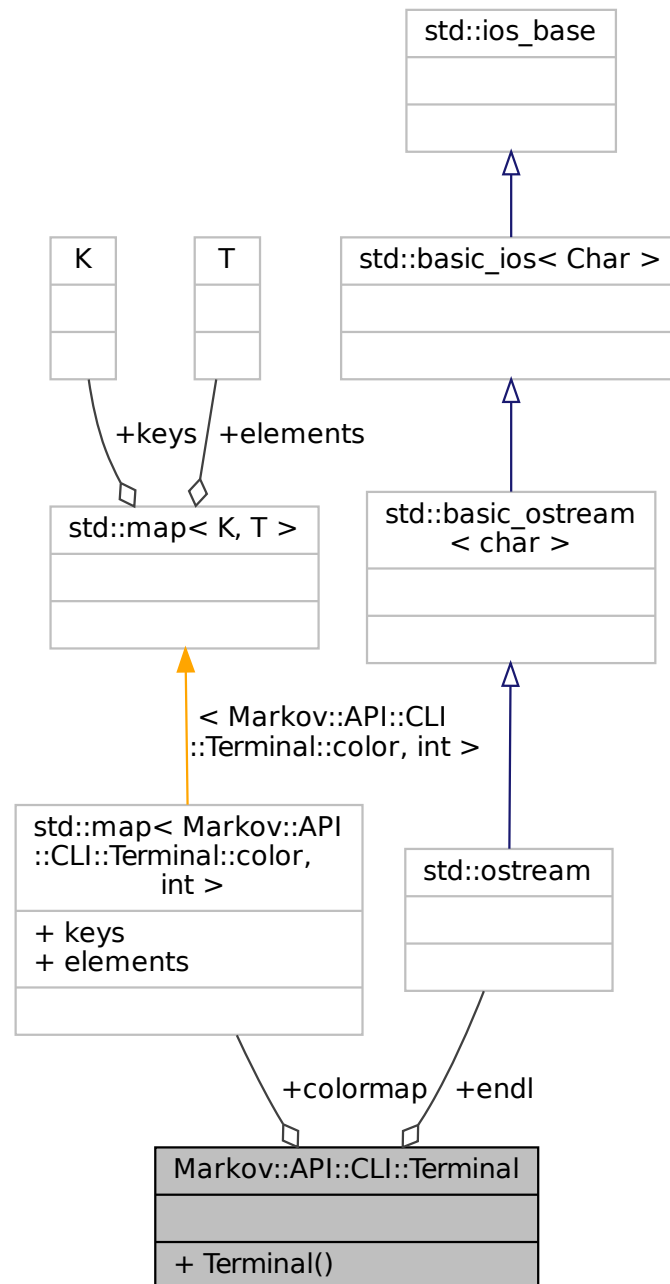
- [random.h](#)

8.18 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](#)

8.18.1 Detailed Description

pretty colors for [Terminal](#). Windows Only.
Definition at line [18](#) of file [term.h](#).

8.18.2 Member Enumeration Documentation

8.18.2.1 color

```
enum Markov::API::CLI::Terminal::color
```

Enumerator

RESET	
BLACK	
RED	
GREEN	
YELLOW	
BLUE	
MAGENTA	
CYAN	
WHITE	
LIGHTGRAY	
DARKGRAY	
BROWN	

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

```
00026 { RESET, BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, LIGHTGRAY, DARKGRAY, BROWN };
```

8.18.3 Constructor & Destructor Documentation

8.18.3.1 Terminal()

```
Terminal::Terminal ( )
```

Default constructor. Get references to stdout and stderr handles.

Definition at line [56](#) of file [term.cpp](#).

```
00056 {  
00057     /*this->*/  
00058 }
```

8.18.4 Member Data Documentation

8.18.4.1 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, 0},  
    {Terminal::color::LIGHTGRAY, 0},  
    {Terminal::color::DARKGRAY, 0},  
    {Terminal::color::YELLOW, 33},  
    {Terminal::color::WHITE, 37},  
    {Terminal::color::RESET, 0},  
}
```

Definition at line 32 of file [term.h](#).

Referenced by [operator<<\(\)](#).

8.18.4.2 endl

```
std::ostream Markov::API::CLI::Terminal::endl [static]
```

Definition at line 37 of file [term.h](#).

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

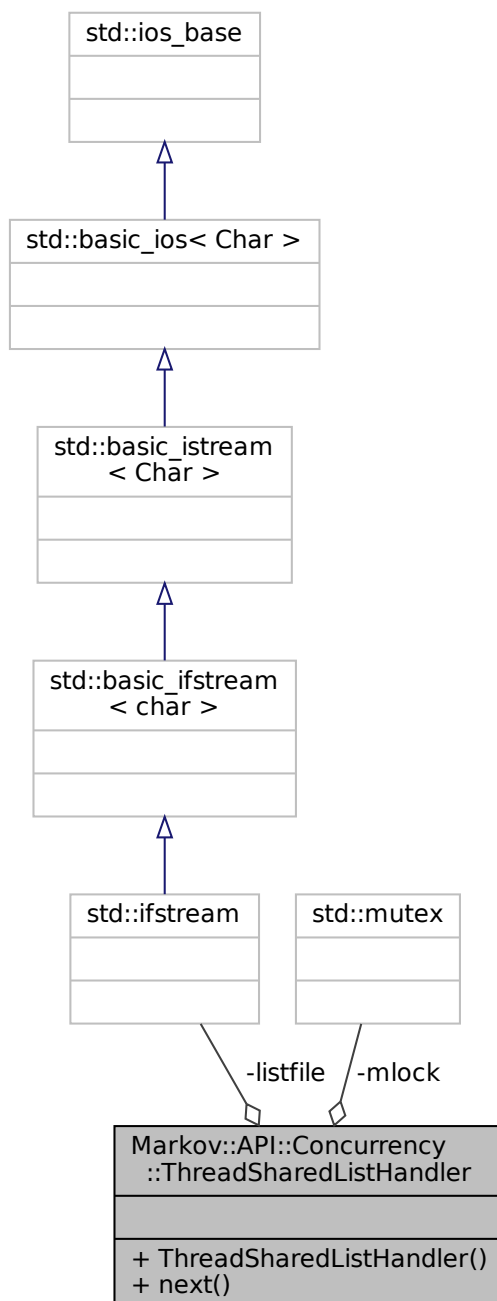
- [term.h](#)
- [term.cpp](#)

8.19 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](#)

8.19.1 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 18 of file [threadSharedListHandler.h](#).

8.19.2 Constructor & Destructor Documentation

8.19.2.1 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

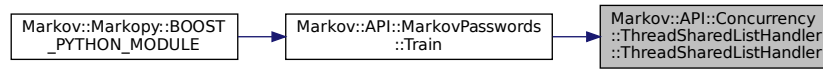
<i>filename</i>	Filename for the file to manage.
-----------------	----------------------------------

Definition at line 4 of file [threadSharedListHandler.cpp](#).

```
00004
00005     this->listfile;
00006     this->listfile.open(filename, std::ios_base::binary);
00007 }
```

References [listfile](#).

Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).
Here is the caller graph for this function:



8.19.3 Member Function Documentation

8.19.3.1 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 10 of file [threadSharedListHandler.cpp](#).

```
00010 {
00011     bool res = false;
00012     this->mlock.lock();
00013     res = (std::getline(this->listfile, *line, '\n')) ? true : false;
00014     this->mlock.unlock();
00015
00016     return res;
00017 }
```

References [listfile](#), and [mlock](#).

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

Here is the caller graph for this function:



8.19.4 Member Data Documentation

8.19.4.1 listfile

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

Definition at line 88 of file [threadSharedListHandler.h](#).

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

8.19.4.2 mlock

```
std::mutex Markov::API::Concurrency::ThreadSharedListHandler::mlock [private]
```

Definition at line 89 of file [threadSharedListHandler.h](#).

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

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

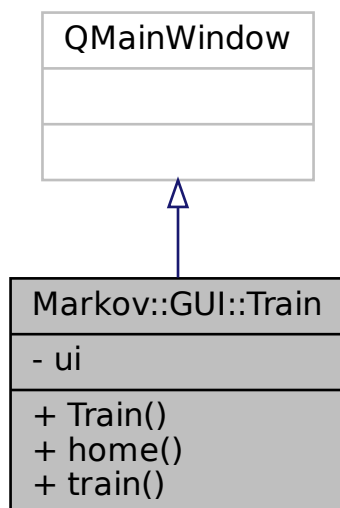
- [threadSharedListHandler.h](#)
- [threadSharedListHandler.cpp](#)

8.20 Markov::GUI::Train Class Reference

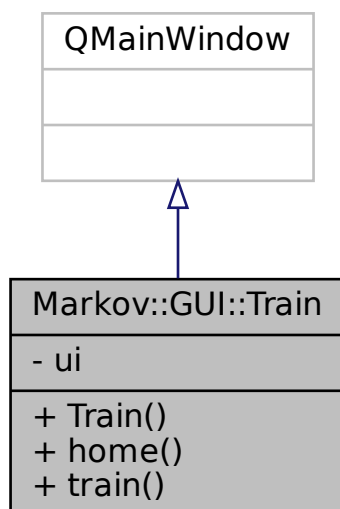
QT Training page class.

```
#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](#)

8.20.1 Detailed Description

QT Training page class.

Definition at line 9 of file [Train.h](#).

8.20.2 Constructor & Destructor Documentation

8.20.2.1 Train()

```
Markov::GUI::Train::Train (  
    QWidget * parent = Q_NULLPTR )
```

8.20.3 Member Function Documentation

8.20.3.1 home

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

8.20.3.2 train

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

8.20.4 Member Data Documentation

8.20.4.1 ui

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

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

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

- [Train.h](#)

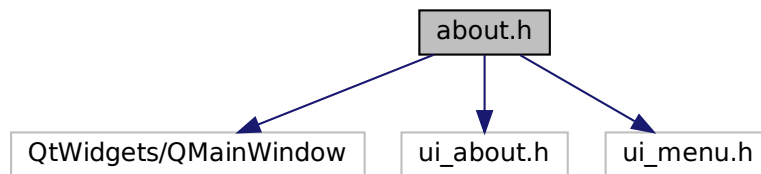
Chapter 9

File Documentation

9.1 about.h File Reference

```
#include <QtWidgets/QMainWindow>
#include "ui_about.h"
#include <ui_menu.h>
```

Include dependency graph for about.h:



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

9.2 about.h

```
00001 #pragma once
00002 #include <QtWidgets/QMainWindow>
00003 #include "ui_about.h"
00004 #include <ui_menu.h>
00005
00006 /** @brief namespace for MarkovPasswords API GUI wrapper
00007 */
00008 namespace Markov::GUI{
00009
00010     /** @brief QT Class for about page
00011     */
00012     class about :public QMainWindow {
```

```

00013     Q_OBJECT
00014     public:
00015         about(QWidget* parent = Q_NULLPTR);
00016
00017     private:
00018         Ui:: main ui;
00019
00020
00021     };
00022 };

```

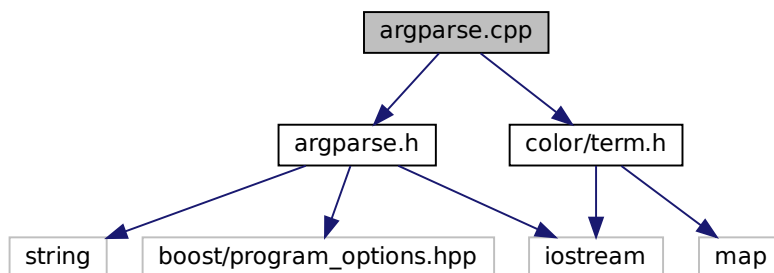
9.3 argparse.cpp File Reference

```

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

```

Include dependency graph for argparse.cpp:



9.4 argparse.cpp

```

00001 #include "argparse.h"
00002 #include "color/term.h"
00003
00004 Markov::API::CLI::ProgramOptions* Markov::API::CLI::Argparse::parse(int argc, char** argv) { return 0;
00005 }
00006
00007
00008 void Markov::API::CLI::Argparse::help() {
00009     std::cout <<
00010     "Markov Passwords - Help\n"
00011     "Options:\n"
00012     "  \n"
00013     "  -of --outputfilename\n"
00014     "      Filename to output the generation results\n"
00015     "  -ef --exportfilename\n"
00016     "      filename to export built model to\n"
00017     "  -if --importfilename\n"
00018     "      filename to import model from\n"
00019     "  -n (generate count)\n"
00020     "      Number of lines to generate\n"
00021     "  \n"
00022     "Usage: \n"
00023     "  markov.exe -if empty_model.mdl -ef model.mdl\n"
00024     "      import empty_model.mdl and train it with data from stdin. When done, output the model to
00025     model.mdl\n"
00026     "  \n"
00027     "  markov.exe -if empty_model.mdl -n 15000 -of wordlist.txt\n"
00028     "      import empty_model.mdl and generate 15000 words to wordlist.txt\n"
00029     << std::endl;
00030 }

```

9.5 argparse.h File Reference

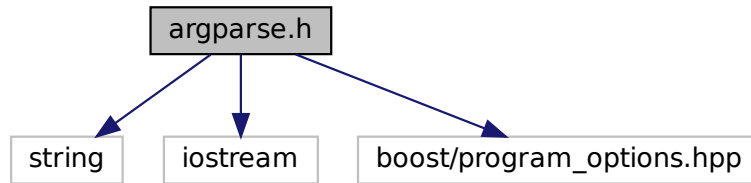
```

#include <string>

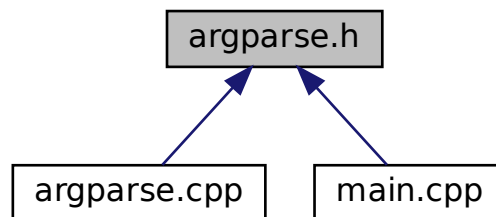
```

```
#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_DYN_LINK](#) 1

Typedefs

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

Structure to hold parsed cli arguments.

9.5.1 Macro Definition Documentation

9.5.1.1 BOOST_ALL_DYN_LINK

```
#define BOOST_ALL_DYN_LINK 1
```

Definition at line 4 of file [argparse.h](#).

9.6 argparse.h

```
00001 #include<string>
00002 #include<iostream>
00003
00004 #define BOOST_ALL_DYN_LINK 1
00005
00006 #include <boost/program_options.hpp>
00007 /** @brief Structure to hold parsed cli arguments.
00008 */
00009 namespace opt = boost::program_options;
00010
00011 /**
00012  @brief Namespace for the CLI objects
00013 */
00014 namespace Markov::API::CLI{
00015
00016     /** @brief Structure to hold parsed cli arguments.
00017     */
00018     typedef struct _programOptions {
00019         bool bImport;
00020         bool bExport;
00021         bool bFailure;
00022         char seperator;
00023         std::string importname;
00024         std::string exportname;
00025         std::string wordlistname;
00026         std::string outputfilename;
00027         std::string datasetname;
00028         int generateN;
00029     } ProgramOptions;
00030
00031
00032     /** @brief Parse command line arguments.
00033     */
00034     class Argparse {
00035     public:
00036
00037         Argparse();
00038
00039         /** @brief Parse command line arguments.
00040         *
00041         * Parses command line arguments to populate ProgramOptions structure.
00042         *
00043         * @param argc Number of command line arguments
00044         * @param argv Array of command line parameters
00045         */
00046         Argparse(int argc, char** argv) {
00047
00048             /*bool bImp;
00049             bool bExp;
00050             bool bFail;
00051             char sprt;
00052             std::string imports;
00053             std::string exports;
00054             std::string outputs;
00055             std::string datasets;
00056             int generateN;
00057             */
00058             opt::options_description desc("Options");
00059
00060
00061             desc.add_options()
00062                 ("generate", "Generate strings with given parameters")
00063                 ("train", "Train model with given parameters")
```

```

00064         ("combine", "Combine")
00065         ("import", opt::value<std::string>(), "Import model file")
00066         ("output", opt::value<std::string>(), "Output model file. This model will be exported
when done. Will be ignored for generation mode")
00067         ("dataset", opt::value<std::string>(), "Dataset file to read input from training. Will
be ignored for generation mode")
00068         ("separator", opt::value<char>(), "Separator character to use with training data.
(character between occurrence and value)")
00069         ("wordlist", opt::value<std::string>(), "Wordlist file path to export generation
results to. Will be ignored for training mode")
00070         ("count", opt::value<int>(), "Number of lines to generate. Ignored in training mode")
00071         ("verbosity", "Output verbosity")
00072         ("help", "Option definitions");
00073
00074     opt::variables_map vm;
00075
00076     opt::store(opt::parse_command_line(argc, argv, desc), vm);
00077
00078     opt::notify(vm);
00079
00080     //std::cout << desc << std::endl;
00081     if (vm.count("help")) {
00082         std::cout << desc << std::endl;
00083     }
00084
00085     if (vm.count("output") == 0) this->po.outputfilename = "NULL";
00086     else if (vm.count("output") == 1) {
00087         this->po.outputfilename = vm["output"].as<std::string>();
00088         this->po.bExport = true;
00089     }
00090     else {
00091         this->po.bFailure = true;
00092         std::cout << "UNIDENTIFIED INPUT" << std::endl;
00093         std::cout << desc << std::endl;
00094     }
00095
00096
00097     if (vm.count("dataset") == 0) this->po.datasetname = "NULL";
00098     else if (vm.count("dataset") == 1) {
00099         this->po.datasetname = vm["dataset"].as<std::string>();
00100     }
00101     else {
00102         this->po.bFailure = true;
00103         std::cout << "UNIDENTIFIED INPUT" << std::endl;
00104         std::cout << desc << std::endl;
00105     }
00106
00107
00108     if (vm.count("wordlist") == 0) this->po.wordlistname = "NULL";
00109     else if (vm.count("wordlist") == 1) {
00110         this->po.wordlistname = vm["wordlist"].as<std::string>();
00111     }
00112     else {
00113         this->po.bFailure = true;
00114         std::cout << "UNIDENTIFIED INPUT" << std::endl;
00115         std::cout << desc << std::endl;
00116     }
00117
00118     if (vm.count("import") == 0) this->po.importname = "NULL";
00119     else if (vm.count("import") == 1) {
00120         this->po.importname = vm["import"].as<std::string>();
00121         this->po.bImport = true;
00122     }
00123     else {
00124         this->po.bFailure = true;
00125         std::cout << "UNIDENTIFIED INPUT" << std::endl;
00126         std::cout << desc << std::endl;
00127     }
00128
00129
00130     if (vm.count("count") == 0) this->po.generateN = 0;
00131     else if (vm.count("count") == 1) {
00132         this->po.generateN = vm["count"].as<int>();
00133     }
00134     else {
00135         this->po.bFailure = true;
00136         std::cout << "UNIDENTIFIED INPUT" << std::endl;
00137         std::cout << desc << std::endl;
00138     }
00139
00140     /*std::cout << vm["output"].as<std::string>() << std::endl;
00141     std::cout << vm["dataset"].as<std::string>() << std::endl;
00142     std::cout << vm["wordlist"].as<std::string>() << std::endl;
00143     std::cout << vm["output"].as<std::string>() << std::endl;
00144     std::cout << vm["count"].as<int>() << std::endl;*/
00145
00146

```

```

00147         //else if (vm.count("train")) std::cout << "train oldu" << std::endl;
00148     }
00149
00150     /** @brief Getter for command line options
00151     *
00152     * Getter for ProgramOptions populated by the argument parser
00153     * @returns ProgramOptions structure.
00154     */
00155     Markov::API::CLI::ProgramOptions getProgramOptions(void) {
00156         return this->po;
00157     }
00158
00159     /** @brief Initialize program options structure.
00160     *
00161     * @param i boolean, true if import operation is flagged
00162     * @param e boolean, true if export operation is flagged
00163     * @param bf boolean, true if there is something wrong with the command line parameters
00164     * @param s separator character for the import function
00165     * @param iName import filename
00166     * @param exName export filename
00167     * @param oName output filename
00168     * @param dName corpus filename
00169     * @param n number of passwords to be generated
00170     *
00171     */
00172     void setProgramOptions(bool i, bool e, bool bf, char s, std::string iName, std::string exName,
std::string oName, std::string dName, int n) {
00173         this->po.bImport = i;
00174         this->po.bExport = e;
00175         this->po.seperator = s;
00176         this->po.bFailure = bf;
00177         this->po.generateN = n;
00178         this->po.importname = iName;
00179         this->po.exportname = exName;
00180         this->po.outputfilename = oName;
00181         this->po.datasetname = dName;
00182
00183         /*strcpy_s(this->po.importname,256,iName);
00184         strcpy_s(this->po.exportname,256,exName);
00185         strcpy_s(this->po.outputfilename,256,oName);
00186         strcpy_s(this->po.datasetname,256,dName);*/
00187     }
00188
00189     /** @brief parse cli commands and return
00190     * @param argc - Program argument count
00191     * @param argv - Program argument values array
00192     * @return ProgramOptions structure.
00193     */
00194     static Markov::API::CLI::ProgramOptions* parse(int argc, char** argv);
00195
00196
00197
00198     /** @brief Print help string.
00199     */
00200     static void help();
00201
00202 private:
00203     Markov::API::CLI::ProgramOptions po;
00204 };
00205
00206 };

```

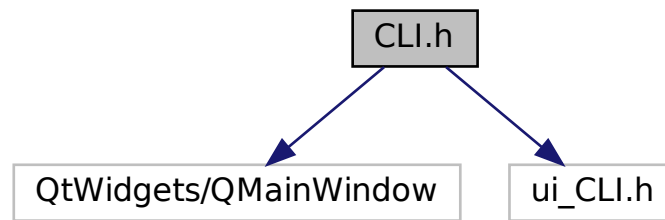
9.7 CLI.h File Reference

```

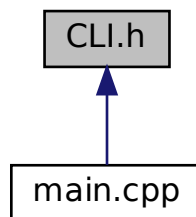
#include <QtWidgets/MainWindow>
#include "ui_CLI.h"

```

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

9.8 CLI.h

```

00001 #pragma once
00002 #include <QtWidgets/QMainWindow>
00003 #include "ui_CLI.h"
00004
00005 namespace Markov::GUI{
00006     /** @brief QT CLI Class
00007     */
00008     class CLI :public QMainWindow {
00009     Q_OBJECT
00010     public:
00011         CLI(QWidget* parent = Q_NULLPTR);
00012 
```

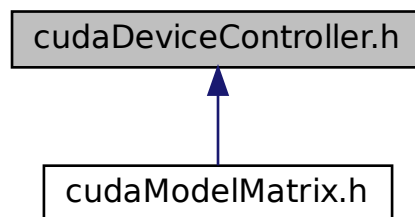
```

00013     private:
00014         Ui::CLI ui;
00015
00016     public slots:
00017         void start();
00018         void statistics();
00019         void about();
00020     };
00021 };

```

9.9 cudaDeviceController.h File Reference

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.

9.10 cudaDeviceController.h

```

00001
00002 /** @brief Namespace for objects requiring CUDA libraries.
00003 */
00004 namespace Markov::API::CUDA{
00005     /** @brief Controller class for CUDA device
00006      *
00007      * This implementation only supports Nvidia devices.
00008      */
00009     class CUDADeviceController{
00010     public:
00011         /** @brief List CUDA devices in the system.
00012          *
00013          * This function will print details of every CUDA capable device in the system.
00014          *
00015          * @b Example @b output:
00016          * @code{.txt}
00017          * Device Number: 0
00018          * Device name: GeForce RTX 2070
00019          * Memory Clock Rate (KHz): 7001000
00020          * Memory Bus Width (bits): 256

```



```

00021      * Peak Memory Bandwidth (GB/s): 448.064
00022      * Max Linear Threads: 1024
00023      * @endcode
00024  */
00025  __host__ void ListCudaDevices();
00026
00027  protected:
00028      /** @brief Check results of the last operation on GPU.
00029      *
00030      * Check the status returned from cudaMalloc/cudaMemcpy to find failures.
00031      *
00032      * If a failure occurs, its assumed beyond redemption, and exited.
00033      * @param _status Cuda error status to check
00034      * @param msg Message to print in case of a failure
00035      * @return 0 if successful, 1 if failure.
00036      * @b Example @b output:
00037      * @code{.txt}
00038      * char *da, a = "test";
00039      * cudastatus = cudaMalloc((char **)&da, 5*sizeof(char*));
00040      * CudaCheckNotifyErr(cudastatus, "Failed to allocate VRAM for *da.\n");
00041      * @endcode
00042      */
00043  __host__ int CudaCheckNotifyErr(cudaError_t _status, const char* msg);
00044
00045  private:
00046      };
00047  };

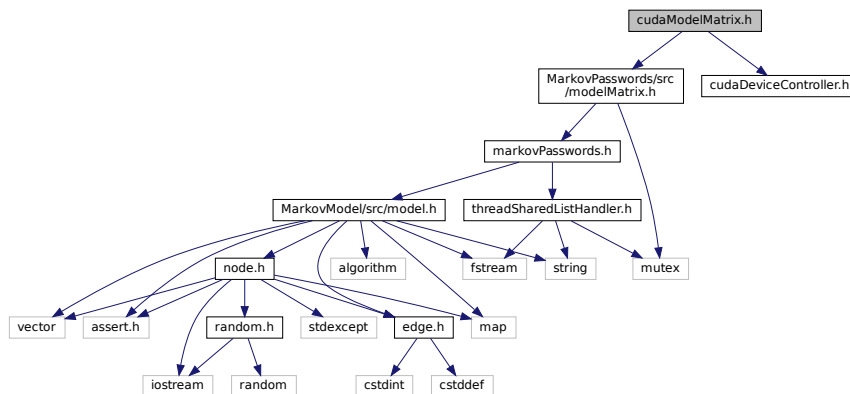
```

9.11 cudaModelMatrix.h File Reference

```
#include "MarkovPasswords/src/modelMatrix.h"
```

```
#include "cudaDeviceController.h"
```

Include dependency graph for cudaModelMatrix.h:



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.

9.12 cudaModelMatrix.h

```

00001 #include "MarkovPasswords/src/modelMatrix.h"
00002 #include "cudaDeviceController.h"
00003
00004 /** @brief Namespace for objects requiring CUDA libraries.
00005 */
00006 namespace Markov::API::CUDA{
00007     /** @brief Extension of Markov::API::ModelMatrix which is modified to run on GPU devices.
00008     *
00009     * This implementation only supports Nvidia devices.
00010     */
00011     class CUDAModelMatrix : public ModelMatrix, public CUDADeviceController{
00012     public:
00013
00014         /** @brief Migrate the class members to the VRAM
00015         *
00016         * Cannot be used without calling Markov::API::ModelMatrix::ConstructMatrix at least once.
00017         * This function will manage the memory allocation and data transfer from CPU RAM to GPU VRAM.
00018         *
00019         * Newly allocated VRAM pointers are set in the class member variables.
00020         *
00021         */
00022         __host__ void MigrateMatrix();
00023
00024         /** @brief Random walk on the Matrix-reduced Markov::Model
00025         *
00026         * TODO
00027         *
00028         *
00029         * @param n - Number of passwords to generate.
00030         * @param wordlistFileName - Filename to write to
00031         * @param minLen - Minimum password length to generate
00032         * @param maxLen - Maximum password length to generate
00033         * @param threads - number of OS threads to spawn
00034         * @param bFileIO - If false, filename will be ignored and will output to stdout.
00035         *
00036         *
00037         * @code{.cpp}
00038         * Markov::API::ModelMatrix mp;
00039         * mp.Import("models/finished.mdl");
00040         * mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
00041         * @endcode
00042         *
00043         */
00044         __host__ void FastRandomWalk(unsigned long int n, const char* wordlistFileName, int minLen,
int maxLen, int threads, bool bFileIO);
00045
00046     protected:
00047         /** @brief Retrieve the result buffer from CUDA kernel.
00048         *
00049         * Done on each partition.
00050         *
00051         *
00052         */
00053         __host__ void RetrieveCudaBuffer(/*TODO*/);
00054
00055     private:
00056         char** device_edgeMatrix;
00057         long int **device_valueMatrix;
00058         char *device_matrixIndex;
00059         long int *device_totalEdgeWeights;
00060     };
00061 };
00062 
```

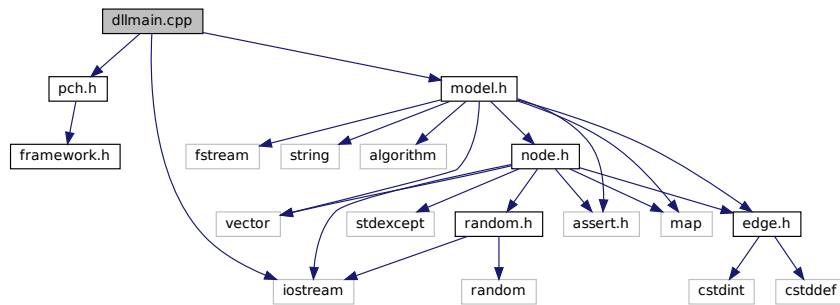
9.13 dllmain.cpp File Reference

```

#include "pch.h"
#include "model.h"
#include <iostream>

```

Include dependency graph for dllmain.cpp:



9.14 dllmain.cpp

```

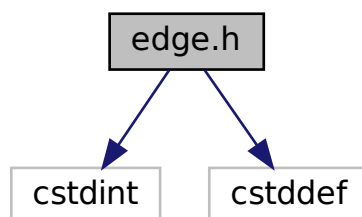
00001 #include "pch.h"
00002 #include "model.h"
00003 #include <iostream>
00004
00005
00006 #ifdef _WIN32
00007 __declspec(dllexport) void dll_loadtest() {
00008     std::cout << "External function called.\n";
00009     //cudaTestEntry();
00010 }
00011
00012 BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)
00013 {
00014     switch (ul_reason_for_call)
00015     {
00016     case DLL_PROCESS_ATTACH:
00017     case DLL_THREAD_ATTACH:
00018     case DLL_THREAD_DETACH:
00019     case DLL_PROCESS_DETACH:
00020         break;
00021     }
00022     return TRUE;
00023 }
00024
00025 #endif
  
```

9.15 edge.h File Reference

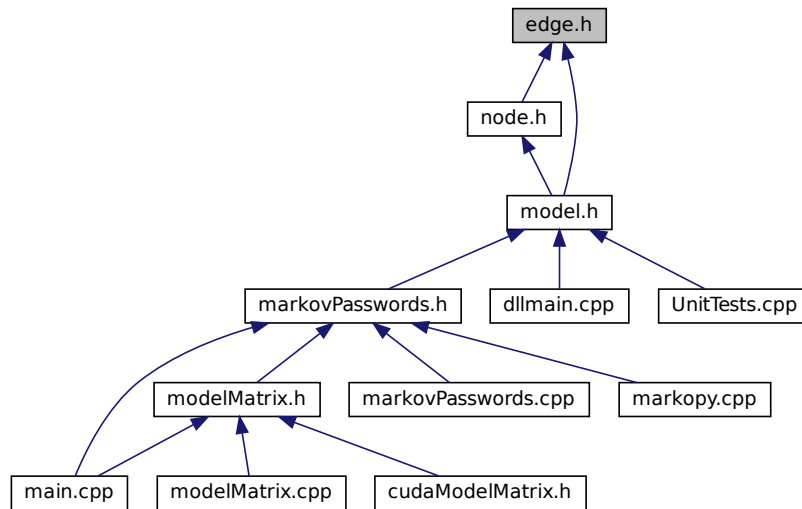
```

#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.

9.16 edge.h

```

00001 #pragma once
00002 #include <stdint>
00003 #include <stddef>
00004
00005 namespace Markov {
00006
00007     template <typename NodeStorageType>
00008     class Node;
00009
00010     /** @brief Edge class used to link nodes in the model together.
00011     *
00012     * Has LeftNode, RightNode, and EdgeWeight of the edge.
00013     * Edges are *UNIDIRECTIONAL* in this model. They can only be traversed LeftNode to RightNode.
00014     */
00015     template <typename NodeStorageType>
00016     class Edge {
00017     public:
00018
00019         /** @brief Default constructor.
00020         */
00021         Edge<NodeStorageType>();
00022
00023         /** @brief Constructor. Initialize edge with given RightNode and LeftNode
00024         * @param _left - Left node of this edge.
00025         * @param _right - Right node of this edge.
00026         *
00027         * @b Example @b Use: Construct edge
00028         * @code{.cpp}
00029         * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');

```

```

00030      * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00031      * Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00032      * @endcode
00033      *
00034      */
00035      Edge<NodeStorageType>(Node<NodeStorageType>* _left, Node<NodeStorageType>* _right);
00036
00037      /** @brief Adjust the edge EdgeWeight with offset.
00038      * Adds the offset parameter to the edge EdgeWeight.
00039      * @param offset - NodeValue to be added to the EdgeWeight
00040      *
00041      * @b Example @b Use: Construct edge
00042      * @code{.cpp}
00043      * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00044      * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00045      * Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00046      *
00047      * e1->AdjustEdge(25);
00048      *
00049      * @endcode
00050      */
00051      void AdjustEdge(long int offset);
00052
00053      /** @brief Traverse this edge to RightNode.
00054      * @return Right node. If this is a terminator node, return NULL
00055      *
00056      *
00057      * @b Example @b Use: Traverse a node
00058      * @code{.cpp}
00059      * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00060      * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00061      * Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00062      *
00063      * e1->AdjustEdge(25);
00064      * Markov::Edge<unsigned char>* e2 = e1->traverseNode();
00065      * @endcode
00066      *
00067      */
00068      inline Node<NodeStorageType>* TraverseNode();
00069
00070      /** @brief Set LeftNode of this edge.
00071      * @param node - Node to be linked with.
00072      */
00073      void SetLeftEdge (Node<NodeStorageType>*);
00074      /** @brief Set RightNode of this edge.
00075      * @param node - Node to be linked with.
00076      */
00077      void SetRightEdge (Node<NodeStorageType>*);
00078
00079      /** @brief return edge's EdgeWeight.
00080      * @return edge's EdgeWeight.
00081      */
00082      inline uint64_t EdgeWeight();
00083
00084      /** @brief return edge's LeftNode
00085      * @return edge's LeftNode.
00086      */
00087      Node<NodeStorageType>* LeftNode();
00088
00089      /** @brief return edge's RightNode
00090      * @return edge's RightNode.
00091      */
00092      inline Node<NodeStorageType>* RightNode();
00093
00094      private:
00095      Node<NodeStorageType>* _left; /** @brief source node*/
00096      Node<NodeStorageType>* _right; /** @brief target node*/
00097      long int _weight; /** @brief Edge EdgeWeight*/
00098      };
00099
00100
00101 };
00102
00103 //default constructor of edge
00104 template <typename NodeStorageType>
00105 Markov::Edge<NodeStorageType>::Edge() {
00106     this->_left = NULL;
00107     this->_right = NULL;
00108     this->_weight = 0;
00109 }
00110 //constructor of edge
00111 template <typename NodeStorageType>
00112 Markov::Edge<NodeStorageType>::Edge(Markov::Node<NodeStorageType>* _left,
    Markov::Node<NodeStorageType>* _right) {
00113     this->_left = _left;
00114     this->_right = _right;
00115     this->_weight = 0;

```

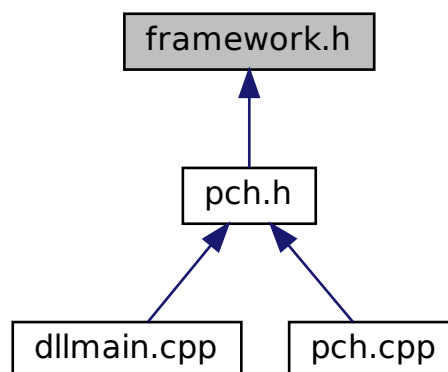
```

00116 }
00117 //to AdjustEdge the edges by the edge with its offset
00118 template <typename NodeStorageType>
00119 void Markov::Edge<NodeStorageType>::AdjustEdge(long int offset) {
00120     this->_weight += offset;
00121     this->LeftNode()->UpdateTotalVerticeWeight(offset);
00122 }
00123 //to TraverseNode the node
00124 template <typename NodeStorageType>
00125 inline Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::TraverseNode() {
00126     if (this->RightNode()->NodeValue() == 0xff) //terminator node
00127         return NULL;
00128     return _right;
00129 }
00130 //to set the LeftNode of the node
00131 template <typename NodeStorageType>
00132 void Markov::Edge<NodeStorageType>::SetLeftEdge(Markov::Node<NodeStorageType>* n) {
00133     this->_left = n;
00134 }
00135 //to set the RightNode of the node
00136 template <typename NodeStorageType>
00137 void Markov::Edge<NodeStorageType>::SetRightEdge(Markov::Node<NodeStorageType>* n) {
00138     this->_right = n;
00139 }
00140 //to get the EdgeWeight of the node
00141 template <typename NodeStorageType>
00142 inline uint64_t Markov::Edge<NodeStorageType>::EdgeWeight() {
00143     return this->_weight;
00144 }
00145 //to get the LeftNode of the node
00146 template <typename NodeStorageType>
00147 Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::LeftNode() {
00148     return this->_left;
00149 }
00150 //to get the RightNode of the node
00151 template <typename NodeStorageType>
00152 inline Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::RightNode() {
00153     return this->_right;
00154 }

```

9.17 framework.h File Reference

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



Macros

- `#define WIN32_LEAN_AND_MEAN`

9.17.1 Macro Definition Documentation

9.17.1.1 WIN32_LEAN_AND_MEAN

```
#define WIN32_LEAN_AND_MEAN
```

Definition at line 3 of file [framework.h](#).

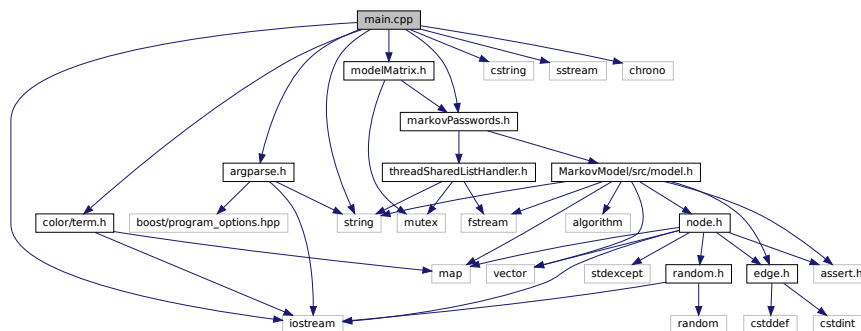
9.18 framework.h

```
00001 #pragma once
00002
00003 #define WIN32_LEAN_AND_MEAN           // Exclude rarely-used stuff from Windows headers
00004 // Windows Header Files
00005
00006 #ifdef _WIN32
00007 #include <windows.h>
00008 #endif
```

9.19 main.cpp File Reference

```
#include <iostream>
#include "color/term.h"
#include "argparse.h"
#include <string>
#include <cstring>
#include <sstream>
#include "markovPasswords.h"
#include "modelMatrix.h"
#include <chrono>
```

Include dependency graph for src/main.cpp:



Functions

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

9.19.1 Function Documentation

9.19.1.1 main()

```
int main (
    int argc,
```

```
char ** argv )
```

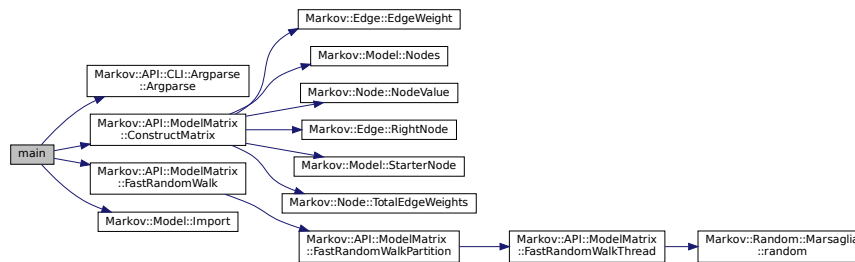
Launch CLI tool.

Definition at line 14 of file [src/main.cpp](#).

```
00014         {
00015
00016     Markov::API::CLI::Terminal t;
00017     /*
00018     ProgramOptions* p = Argparse::parse(argc, argv);
00019
00020     if (p==0 || p->bFailure) {
00021         std::cout << TERM_FAIL << "Arguments Failed to Parse" << std::endl;
00022         Argparse::help();
00023     }*/
00024     Markov::API::CLI::Argparse a(argc,argv);
00025
00026     Markov::API::ModelMatrix markovPass;
00027     std::cerr << "Importing model.\n";
00028     markovPass.Import("models/finished.mdl");
00029     std::cerr << "Import done. \n";
00030     markovPass.ConstructMatrix();
00031     std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
00032     //markovPass.FastRandomWalk(500000000,"/media/ignis/Stuff/wordlist.txt",6,12,25, true);
00033     markovPass.FastRandomWalk(500000000,"/media/ignis/Stuff/wordlist2.txt",6,12,25, true);
00034     std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
00035
00036     std::cerr << "Finished in:" << std::chrono::duration_cast<std::chrono::milliseconds>(end -
00037     begin).count() << " milliseconds" << std::endl;
00038     return 0;
00039 }
```

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

Here is the call graph for this function:



9.20 src/main.cpp

```
00001 #pragma once
00002 #include <iostream>
00003 #include "color/term.h"
00004 #include "argparse.h"
00005 #include <string>
00006 #include <cstring>
00007 #include <sstream>
00008 #include "markovPasswords.h"
00009 #include "modelMatrix.h"
00010 #include <chrono>
00011
00012 /** @brief Launch CLI tool.
00013 */
00014 int main(int argc, char** argv) {
00015
00016     Markov::API::CLI::Terminal t;
00017     /*
00018     ProgramOptions* p = Argparse::parse(argc, argv);
00019
00020     if (p==0 || p->bFailure) {
00021         std::cout << TERM_FAIL << "Arguments Failed to Parse" << std::endl;
00022         Argparse::help();
00023     }*/
00024     Markov::API::CLI::Argparse a(argc,argv);
00025
00026     Markov::API::ModelMatrix markovPass;
00027     std::cerr << "Importing model.\n";
00028     markovPass.Import("models/finished.mdl");
00029     std::cerr << "Import done. \n";
```



```

00030     markovPass.ConstructMatrix();
00031     std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
00032     //markovPass.FastRandomWalk(50000000,"/media/ignis/Stuff/wordlist.txt",6,12,25, true);
00033     markovPass.FastRandomWalk(500000000,"/media/ignis/Stuff/wordlist2.txt",6,12,25, true);
00034     std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
00035
00036     std::cerr << "Finished in:" << std::chrono::duration_cast<std::chrono::milliseconds> (end -
begin).count() << " milliseconds" << std::endl;
00037     return 0;
00038 }

```

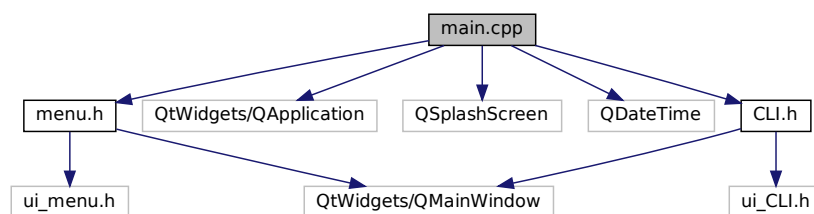
9.21 main.cpp File Reference

```

#include "menu.h"
#include <QtWidgets/QApplication>
#include <QSplashScreen>
#include <QDateTime>
#include "CLI.h"

```

Include dependency graph for UI/src/main.cpp:



Functions

- int [main](#) (int argc, char *argv[])
Launch UI.

9.21.1 Function Documentation

9.21.1.1 main()

```

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

```

Launch UI.

Definition at line 12 of file [UI/src/main.cpp](#).

```

00013 {
00014
00015
00016
00017     QApplication a(argc, argv);
00018
00019     QPixmap loadingPix("views/startup.jpg");
00020     QSplashScreen splash(loadingPix);
00021     splash.show();
00022     QDateTime time = QDateTime::currentDateTime();
00023     QDateTime currentTime = QDateTime::currentDateTime(); //Record current time
00024     while (time.secsTo(currentTime) <= 5) //5 is the number of seconds to delay
00025     {
00026         currentTime = QDateTime::currentDateTime();
00027         a.processEvents();
00028     };
00029
00030
00031     CLI w;

```

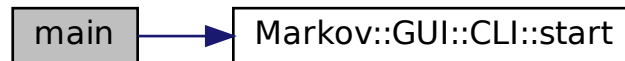
```

00032     w.show();
00033     splash.finish(&w);
00034     return a.exec();
00035 }

```

References [Markov::GUI::CLI::start\(\)](#).

Here is the call graph for this function:



9.22 UI/src/main.cpp

```

00001 // #include "MarkovPasswordsGUI.h"
00002 #include "menu.h"
00003 #include <QtWidgets/QApplication>
00004 #include <QSplashScreen>
00005 #include < QDateTime >
00006 #include "CLI.h"
00007
00008 using namespace Markov::GUI;
00009
00010 /** @brief Launch UI.
00011 */
00012 int main(int argc, char *argv[])
00013 {
00014
00015
00016     QApplication a(argc, argv);
00017
00018     QPixmap loadingPix("views/startup.jpg");
00019     QSplashScreen splash(loadingPix);
00020     splash.show();
00021     QDateTime time = QDateTime::currentDateTime();
00022     QDateTime currentTime = QDateTime::currentDateTime(); //Record current time
00023     while (time.secsTo(currentTime) <= 5) //5 is the number of seconds to delay
00024     {
00025         currentTime = QDateTime::currentDateTime();
00026         a.processEvents();
00027     };
00028
00029
00030     CLI w;
00031     w.show();
00032     splash.finish(&w);
00033     return a.exec();
00034 }
00035 }

```

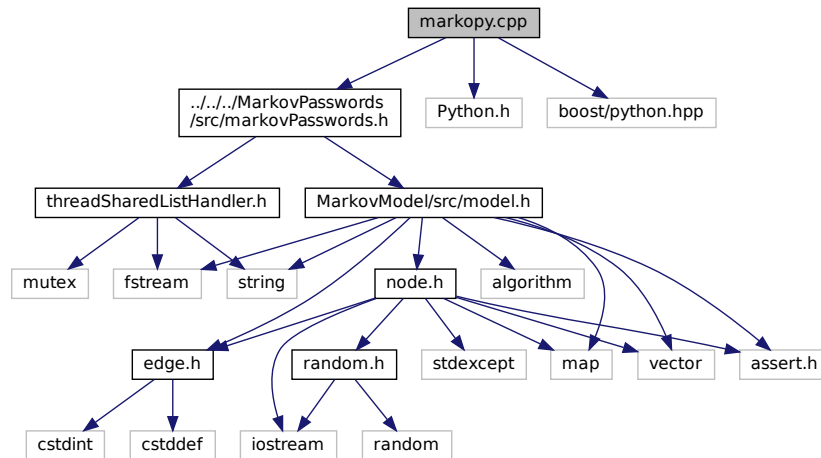
9.23 markopy.cpp File Reference

```

#include "../.../MarkovPasswords/src/markovPasswords.h"
#include <Python.h>
#include <boost/python.hpp>

```

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_PYTHON_STATIC_LIB`

Functions

- [Markov::Markopy::BOOST_PYTHON_MODULE](#) (markopy)

9.23.1 Macro Definition Documentation

9.23.1.1 BOOST_PYTHON_STATIC_LIB

```
#define BOOST_PYTHON_STATIC_LIB
```

Definition at line 4 of file [markopy.cpp](#).

9.24 markopy.cpp

```

00001 #pragma once
00002 #include "../../MarkovPasswords/src/markovPasswords.h"
00003
00004 #define BOOST_PYTHON_STATIC_LIB
00005 #include <Python.h>
00006 #include <boost/python.hpp>
00007
00008 using namespace boost::python;
00009
00010 namespace Markov::Markopy{
00011     BOOST_PYTHON_MODULE(markopy)
00012     {
00013         bool (Markov::API::MarkovPasswords::*Import) (const char*) = &Markov::Model<char>::Import;
00014         bool (Markov::API::MarkovPasswords::*Export) (const char*) = &Markov::Model<char>::Export;
00015         class_<Markov::API::MarkovPasswords>("MarkovPasswords", init<>())
00016             .def(init<>())
  
```

```

00017         .def("Train", &Markov::API::MarkovPasswords::Train,
00018             "Train the model\n"
00019             "\n"
00020             ":param datasetFileName: Ifstream* to the dataset. If null, use class member\n"
00021             ":param delimiter: a character, same as the delimiter in dataset content\n"
00022             ":param threads: number of OS threads to spawn\n")
00023         .def("Generate", &Markov::API::MarkovPasswords::Generate,
00024             "Generate passwords from a trained model.\n"
00025             ":param n: Ifstream* to the dataset. If null, use class member\n"
00026             ":param wordlistFileName: a character, same as the delimiter in dataset content\n"
00027             ":param minLen: number of OS threads to spawn\n"
00028             ":param maxLen: Ifstream* to the dataset. If null, use class member\n"
00029             ":param threads: a character, same as the delimiter in dataset content\n"
00030             ":param threads: number of OS threads to spawn\n")
00031         .def("Import", Import, "Import a model file.")
00032         .def("Export", Export, "Export a model to file.")
00033     ;
00034 };
00035 };

```

9.25 markopy_cli.py File Reference

Namespaces

- [markopy_cli](#)

Functions

- def [markopy_cli.cli_init](#) (input_model)
- def [markopy_cli.cli_train](#) (model, dataset, separator, output, output_forced=False, bulk=False)
- def [markopy_cli.cli_generate](#) (model, wordlist, bulk=False)

Variables

- [markopy_cli.parser](#)
- [markopy_cli.help](#)
- [markopy_cli.default](#)
- [markopy_cli.action](#)
- [markopy_cli.args](#) = parser.parse_args()
- [markopy_cli.corpus_list](#) = os.listdir(args.dataset)
- def [markopy_cli.model](#) = cli_init(args.input)
- [markopy_cli.output_file_name](#) = corpus
- string [markopy_cli.model_extension](#) = ""
- [markopy_cli.output_forced](#)
- [markopy_cli.True](#)
- [markopy_cli.bulk](#)
- [markopy_cli.model_list](#) = os.listdir(args.input)
- [markopy_cli.model_base](#) = input
- [markopy_cli.output](#)

9.26 markopy_cli.py

```

00001 #!/usr/bin/python3
00002 """
00003     @namespace Markov::Markopy::Python
00004 """
00005
00006 import markopy
00007 import argparse
00008 import allocate as logging
00009 import re
00010 import os
00011
00012 parser = argparse.ArgumentParser(description="Python wrapper for MarkovPasswords.",
00013                                 epilog=f"""Sample runs:
00014 {__file__} train untrained.mdl -d dataset.dat -s "\t" -o trained.mdl

```

```

00015     Import untrained.mdl, train it with dataset.dat which has tab delimited data, output resulting
00016     model to trained.mdl\n
00017 {__file__} generate trained.mdl -n 500 -w output.txt
00018     Import trained.mdl, and generate 500 lines to output.txt
00019
00020 {__file__} combine untrained.mdl -d dataset.dat -s "\\t" -n 500 -w output.txt
00021     Train and immediately generate 500 lines to output.txt. Do not export trained model.
00022
00023 {__file__} combine untrained.mdl -d dataset.dat -s "\\t" -n 500 -w output.txt -o trained.mdl
00024     Train and immediately generate 500 lines to output.txt. Export trained model.
00025 """", formatter_class=argparse.RawTextHelpFormatter)
00026
00027 parser.add_argument("mode", help="Operation mode, supported modes: \"generate\", \"train\" and
00028     \"combine\".")
00029 parser.add_argument("input", help="Input model file. This model will be imported before starting
00030     operation.\n"
00031         + "For more information on the file structure for input, check out
00032     the wiki page.")
00033 parser.add_argument("-o", "--output",
00034     help="Output model filename. This model will be exported when done. Will be
00035     ignored for generation mode.")
00036 parser.add_argument("-d", "--dataset",
00037     help="Dataset filename to read input from for training. Will be ignored for
00038     generation mode.\n"
00039     + "Dataset is occurrence of a string and the string value seperated by a
00040     seperator. For more info "
00041     + "on the dataset file structure, check out the github wiki page.")
00042 parser.add_argument("-s", "--seperator",
00043     help="Seperator character to use with training data.(character between occurrence
00044     and value)\n"
00045     + "For more information on dataset/corpus file structure, check out the github
00046     wiki.")
00047 parser.add_argument("-w", "--wordlist",
00048     help="Wordlist filename path to export generation results to. Will be ignored for
00049     training mode")
00050 parser.add_argument("--min", default=6, help="Minimum length that is allowed during generation.\n"
00051     + "Any string shorter than this parameter will retry to continue instead of
00052     proceeding to "
00053     + "finishing node")
00054 parser.add_argument("--max", default=12, help="Maximum length that is allowed during generation.\n"
00055     + "Any string that does reaches this length are cut off irregardless to their
00056     position on the model.")
00057 parser.add_argument("-n", "--count", help="Number of lines to generate. Ignored in training mode.")
00058 parser.add_argument("-t", "--threads", default=10, help="Number of threads to use with
00059     training/generation.\n"
00060     + "This many OS threads will be created for training/generation functions")
00061 parser.add_argument("-v", "--verbosity", action="count", help="Output verbosity.\n"
00062     + "Set verbosity to 1: -v\n"
00063     + "Set verbosity to 3: -vvv\n"
00064     + "Print pretty much everything, including caller functions: -vvvvvvvvvvvvvvvv")
00065 parser.add_argument("-b", "--bulk", action="store_true",
00066     help="Bulk generate or bulk train every corpus/model in the folder.\n"
00067     + "If working on this mode, output/input/dataset parameters should be a folder.\n"
00068     + "Selected operation (generate/train) will be applied to each file in the folder,
00069     and "
00070     + "output to the output directory.")
00071 args = parser.parse_args()
00072
00073
00074 def cli_init(input_model):
00075     logging.VERBOSITY = 0
00076     if args.verbosity:
00077         logging.VERBOSITY = args.verbosity
00078         logging.pprint(f"Verbosity set to {args.verbosity}.", 2)
00079
00080     logging.pprint("Initializing model.", 1)
00081     model = markopy.MarkovPasswords()
00082     logging.pprint("Model initialized.", 2)
00083
00084     logging.pprint("Importing model file.", 1)
00085
00086     if (not os.path.isfile(input_model)):
00087         logging.pprint(f"Model file at {input_model} not found. Check the file path, or working
00088         directory")
00089         exit(1)
00090
00091     model.Import(input_model)
00092     logging.pprint("Model imported successfully.", 2)
00093     return model
00094
00095 def cli_train(model, dataset, seperator, output, output_forced=False, bulk=False):
00096     if not (dataset and seperator and (output or not output_forced)):
00097         logging.pprint(
00098             f"Training mode requires -d/--dataset{' ', -o/--output' if output_forced else ''} and
00099             -s/--seperator parameters. Exiting.")

```

```

00086         exit(2)
00087
00088     if (not bulk and not os.path.isfile(dataset)):
00089         logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00090         exit(3)
00091
00092     if (output and os.path.isfile(output)):
00093         logging.pprint(f"{output} exists and will be overwritten.", 1)
00094
00095     if (separator == '\\t'):
00096         logging.pprint("Escaping seperator.", 3)
00097         separator = '\t'
00098
00099     if (len(separator) != 1):
00100         logging.pprint(f"Delimiter must be a single character, and \"{separator}\" is not accepted.")
00101         exit(4)
00102
00103     logging.pprint(f'Starting training.', 3)
00104     model.Train(dataset, separator, int(args.threads))
00105     logging.pprint(f'Training completed.', 2)
00106
00107     if (output):
00108         logging.pprint(f'Exporting model to {output}', 2)
00109         model.Export(output)
00110     else:
00111         logging.pprint(f'Model will not be exported.', 1)
00112
00113
00114 def cli_generate(model, wordlist, bulk=False):
00115     if not (wordlist or args.count):
00116         logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters. Exiting.")
00117         exit(2)
00118
00119     if (bulk and os.path.isfile(wordlist)):
00120         logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00121     model.Generate(int(args.count), wordlist, int(args.min), int(args.max), int(args.threads))
00122
00123
00124 if (args.bulk):
00125     logging.pprint(f"Bulk mode operation chosen.", 4)
00126
00127     if (args.mode.lower() == "train"):
00128         if (os.path.isdir(args.output) and not os.path.isfile(args.output)) and (
00129             os.path.isdir(args.dataset) and not os.path.isfile(args.dataset)):
00130             corpus_list = os.listdir(args.dataset)
00131             for corpus in corpus_list:
00132                 model = cli_init(args.input)
00133                 logging.pprint(f"Training {args.input} with {corpus}", 2)
00134                 output_file_name = corpus
00135                 model_extension = ""
00136                 if "." in args.input:
00137                     model_extension = args.input.split(".")[1]
00138                 cli_train(model, f"{args.dataset}/{corpus}", args.seperator,
00139                     f"{args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00140             else:
00141                 logging.pprint("In bulk training, output and dataset should be a directory.")
00142                 exit(1)
00143
00144     elif (args.mode.lower() == "generate"):
00145         if (os.path.isdir(args.wordlist) and not os.path.isfile(args.wordlist)) and (
00146             os.path.isdir(args.input) and not os.path.isfile(args.input)):
00147             model_list = os.listdir(args.input)
00148             print(model_list)
00149             for input in model_list:
00150                 logging.pprint(f"Generating from {args.input}/{input} to {args.wordlist}/{input}.txt",
00151                     2)
00152
00153                 model = cli_init(f"{args.input}/{input}")
00154                 model_base = input
00155                 if "." in args.input:
00156                     model_base = input.split(".")[1]
00157                 cli_generate(model, f"{args.wordlist}/{model_base}.txt", bulk=True)
00158             else:
00159                 logging.pprint("In bulk generation, input and wordlist should be directory.")
00160
00161     else:
00162         model = cli_init(args.input)
00163         if (args.mode.lower() == "generate"):
00164             cli_generate(model, args.wordlist)
00165
00166     elif (args.mode.lower() == "train"):
00167         cli_train(model, args.dataset, args.seperator, args.output, output_forced=True)
00168
00169     elif (args.mode.lower() == "combine"):
00170         cli_train(model, args.dataset, args.seperator, args.output)

```

```

00172         cli_generate(model, args.wordlist)
00173
00174
00175     else:
00176         logging.pprint("Invalid mode argument given.")
00177         logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00178         exit(5)

```

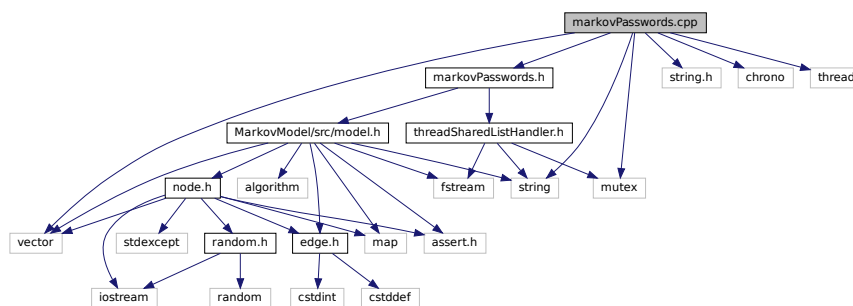
9.27 markovPasswords.cpp File Reference

```

#include "markovPasswords.h"
#include <string.h>
#include <chrono>
#include <thread>
#include <vector>
#include <mutex>
#include <string>

```

Include dependency graph for markovPasswords.cpp:



9.28 markovPasswords.cpp

```

00001 #pragma once
00002 #include "markovPasswords.h"
00003 #include <string.h>
00004 #include <chrono>
00005 #include <thread>
00006 #include <vector>
00007 #include <mutex>
00008 #include <string>
00009
00010 Markov::API::MarkovPasswords::MarkovPasswords() : Markov::Model<char>() {
00011
00012
00013 }
00014
00015 Markov::API::MarkovPasswords::MarkovPasswords(const char* filename) {
00016
00017     std::ifstream* importFile;
00018
00019     this->Import(filename);
00020
00021     //std::ifstream* newFile(filename);
00022
00023     //importFile = newFile;
00024
00025 }
00026
00027 std::ifstream* Markov::API::MarkovPasswords::OpenDatasetFile(const char* filename) {
00028
00029     std::ifstream* datasetFile;
00030
00031     std::ifstream newFile(filename);
00032
00033     datasetFile = &newFile;
00034
00035     this->Import(datasetFile);
00036     return datasetFile;

```

```

00037 }
00038
00039
00040 void Markov::API::MarkovPasswords::Train(const char* datasetFileName, char delimiter, int threads) {
00041     Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
00042     auto start = std::chrono::high_resolution_clock::now();
00043
00044     std::vector<std::thread*> threadsV;
00045     for(int i=0;i<threads;i++){
00046         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
00047             &listhandler, delimiter));
00048     }
00049
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 Markov::API::MarkovPasswords::TrainThread(Markov::API::Concurrency::ThreadSharedListHandler
00060     *listhandler, 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 #ifdef _WIN32
00071         sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
00072 #else
00073         sscanf(line.c_str(), format_str, &oc, linebuf);
00074 #endif
00075         this->AdjustEdge((const char*)linebuf, oc);
00076         delete linebuf;
00077     }
00078
00079
00080 std::ofstream* Markov::API::MarkovPasswords::Save(const char* filename) {
00081     std::ofstream* exportFile;
00082
00083     std::ofstream newFile(filename);
00084
00085     exportFile = &newFile;
00086
00087     this->Export(exportFile);
00088     return exportFile;
00089 }
00090
00091
00092 void Markov::API::MarkovPasswords::Generate(unsigned long int n, const char* wordlistFileName, int
00093     minLen, int maxLen, int threads) {
00094     char* res;
00095     char print[100];
00096     std::ofstream wordlist;
00097     wordlist.open(wordlistFileName);
00098     std::mutex mlock;
00099     int iterationsPerThread = n/threads;
00100     int iterationsCarryOver = n%threads;
00101     std::vector<std::thread*> threadsV;
00102     for(int i=0;i<threads;i++){
00103         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00104             &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00105     }
00106
00107     for(int i=0;i<threads;i++){
00108         threadsV[i]->join();
00109         delete threadsV[i];
00110     }
00111
00112     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00113
00114 void Markov::API::MarkovPasswords::GenerateThread(std::mutex *outputLock, unsigned long int n,
00115     std::ofstream *wordlist, int minLen, int maxLen) {
00116     char* res = new char[maxLen+5];
00117     if(n==0) return;
00118
00119     Markov::Random::Marsaglia MarsagliaRandomEngine;

```



```

00119     for (int i = 0; i < n; i++) {
00120         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00121         outputLock->lock();
00122         *wordlist « res « "\n";
00123         outputLock->unlock();
00124     }
00125 }

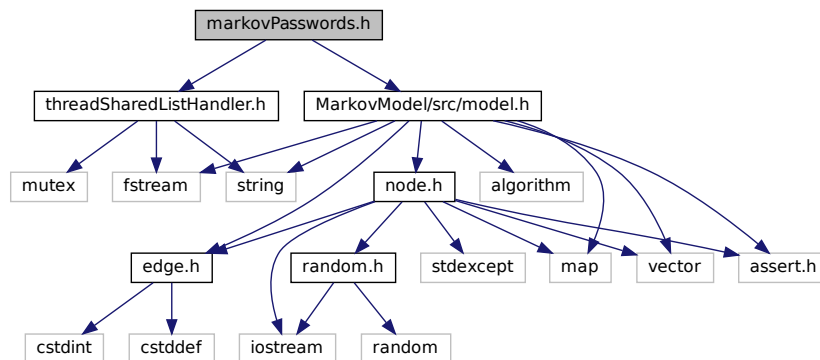
```

9.29 markovPasswords.h File Reference

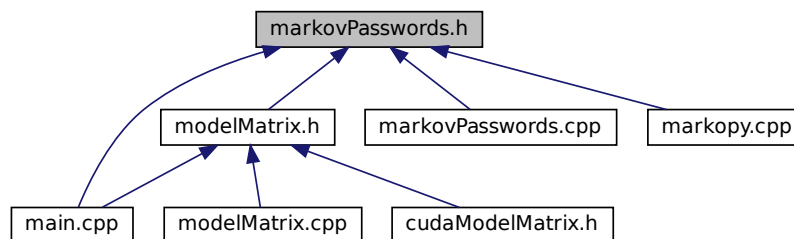
```
#include "threadSharedListHandler.h"
```

```
#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.

9.30 markovPasswords.h

```

00001 #pragma once
00002 #include "threadSharedListHandler.h"
00003 #include "MarkovModel/src/model.h"
00004
00005
00006 /** @brief Namespace for the MarkovPasswords API
00007 */
00008 namespace Markov::API{
00009
00010     /** @brief Markov::Model with char represented nodes.
00011     *
00012     * Includes wrappers for Markov::Model and additional helper functions to handle file I/O
00013     *
00014     * This class is an extension of Markov::Model<char>, with higher level abstractions such as train
    and generate.
00015     *
00016     */
00017     class MarkovPasswords : public Markov::Model<char>{
00018     public:
00019
00020         /** @brief Initialize the markov model from MarkovModel::Markov::Model.
00021         *
00022         * Parent constructor. Has no extra functionality.
00023         */
00024         MarkovPasswords();
00025
00026         /** @brief Initialize the markov model from MarkovModel::Markov::Model, with an import file.
00027         *
00028         * This function calls the Markov::Model::Import on the filename to construct the model.
00029         * Same thing as creating an empty model, and calling MarkovPasswords::Import on the
    filename.
00030         *
00031         * @param filename - Filename to import
00032         *
00033         * @b Example @b Use: Construction via filename
00034         * @code{.cpp}
00035         * MarkovPasswords mp("test.mdl");
00036         * @endcode
00037         */
00038         MarkovPasswords(const char* filename);
00039
00040
00041         /** @brief Open dataset file and return the ifstream pointer
00042         * @param filename - Filename to open
00043         * @return ifstream* to the dataset file
00044         */
00045         std::ifstream* OpenDatasetFile(const char* filename);
00046
00047
00048         /** @brief Train the model with the dataset file.
00049         * @param datasetFileName - Ifstream* to the dataset. If null, use class member
00050         * @param delimiter - a character, same as the delimiter in dataset content
00051         * @param threads - number of OS threads to spawn
00052         *
00053         * @code{.cpp}
00054         * Markov::API::MarkovPasswords mp;
00055         * mp.Import("models/2gram.mdl");
00056         * mp.Train("password.corpus");
00057         * @endcode
00058         */
00059         void Train(const char* datasetFileName, char delimiter, int threads);
00060
00061
00062
00063         /** @brief Export model to file.
00064         * @param filename - Export filename.
00065         * @return std::ofstream* of the exported file.
00066         */
00067         std::ofstream* Save(const char* filename);
00068
00069         /** @brief Call Markov::Model::RandomWalk n times, and collect output.
00070         *
00071         * Generate from model and write results to a file.
00072         * a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5
    on average.
00073         *
00074         * @deprecated See Markov::API::MatrixModel::FastRandomWalk for more information.
00075         * @param n - Number of passwords to generate.
00076         * @param wordlistFileName - Filename to write to
00077         * @param minLen - Minimum password length to generate
00078         * @param maxLen - Maximum password length to generate
00079         * @param threads - number of OS threads to spawn
00080         */
00081         void Generate(unsigned long int n, const char* wordlistFileName, int minLen=6, int maxLen=12,
    int threads=20);

```

```

00082
00083
00084     private:
00085
00086         /** @brief A single thread invoked by the Train function.
00087          * @param listhandler - Listhandler class to read corpus from
00088          * @param delimiter - a character, same as the delimiter in dataset content
00089          */
00090         void TrainThread(Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char
00091 delimiter);
00092
00093         /** @brief A single thread invoked by the Generate function.
00094          *
00095          * @b DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information.
00096          * This has been replaced with
00097          * a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5
00098          * on average.
00099          * @param outputLock - shared mutex lock to lock during output operation. Prevents race
00100          * condition on write.
00101          * @param n number of lines to be generated by this thread
00102          * @param wordlist wordlistfile
00103          * @param minLen - Minimum password length to generate
00104          * @param maxLen - Maximum password length to generate
00105          */
00106         void GenerateThread(std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int
00107 minLen, int maxLen);
00108         std::ifstream* datasetFile;
00109         std::ofstream* modelSavefile;
00110         std::ofstream* outputFile;
00111     };
00112
00113 };

```

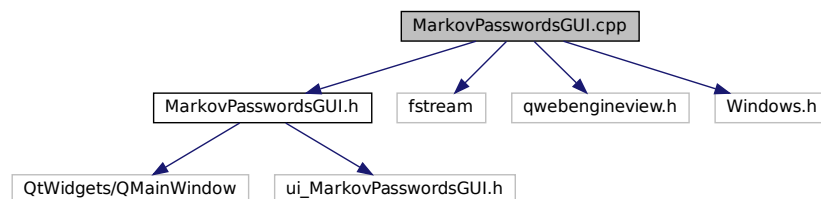
9.31 MarkovPasswordsGUI.cpp File Reference

```

#include "MarkovPasswordsGUI.h"
#include <fstream>
#include <qwebengineview.h>
#include <Windows.h>

```

Include dependency graph for MarkovPasswordsGUI.cpp:



9.32 MarkovPasswordsGUI.cpp

```

00001 #include "MarkovPasswordsGUI.h"
00002 #include <fstream>
00003 #include <qwebengineview.h>
00004 #include <Windows.h>
00005
00006 using namespace Markov::GUI;
00007
00008 MarkovPasswordsGUI::MarkovPasswordsGUI(QWidget *parent)
00009     : QMainWindow(parent)
00010 {
00011     ui.setupUi(this);
00012
00013     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {benchmarkSelected(); });
00014

```

```

00015     QObject::connect(ui.pushButton_2,&QPushButton::clicked, this, [this] {modelvisSelected(); });
00016     QObject::connect(ui.pushButton_4, &QPushButton::clicked, this, [this] {comparisonSelected(); });
00017 }
00018
00019
00020 /*
00021 Methods for buttons
00022 */
00023
00024 void MarkovPasswordsGUI::benchmarkSelected() {
00025
00026     QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00027
00028     //get working directory
00029     char path[255];
00030     GetCurrentDirectoryA(255, path);
00031
00032     //get absolute path to the layout html
00033     std::string layout = "file:///\" + std::string(path) + "\\views\\example.html";
00034     std::replace(layout.begin(), layout.end(), '\\', '/');
00035     webkit->setUrl(QUrl(layout.c_str()));
00036 }
00037
00038
00039 void MarkovPasswordsGUI::modelvisSelected() {
00040
00041     QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00042
00043     //get working directory
00044     char path[255];
00045     GetCurrentDirectoryA(255, path);
00046
00047     //get absolute path to the layout html
00048     std::string layout = "file:///\" + std::string(path) + "\\views\\model.htm";
00049     std::replace(layout.begin(), layout.end(), '\\', '/');
00050     webkit->setUrl(QUrl(layout.c_str()));
00051 }
00052
00053 void MarkovPasswordsGUI::comparisonSelected() {
00054
00055     QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00056
00057     //get working directory
00058     char path[255];
00059     GetCurrentDirectoryA(255, path);
00060
00061     //get absolute path to the layout html
00062     std::string layout = "file:///\" + std::string(path) + "\\views\\comparison.htm";
00063     std::replace(layout.begin(), layout.end(), '\\', '/');
00064     webkit->setUrl(QUrl(layout.c_str()));
00065 }
00066
00067
00068
00069
00070
00071 void MarkovPasswordsGUI::renderHTMLFile(std::string* filename) {
00072     //extract and parametrize the code from constructor
00073 }
00074 }
00075
00076
00077
00078 void MarkovPasswordsGUI::loadDataset(std::string* filename) {
00079     //extract and parametrize the code from constructor
00080 }
00081 }

```

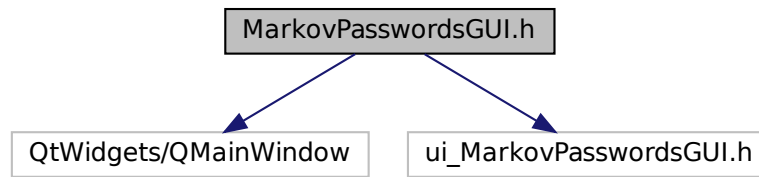
9.33 MarkovPasswordsGUI.h File Reference

```

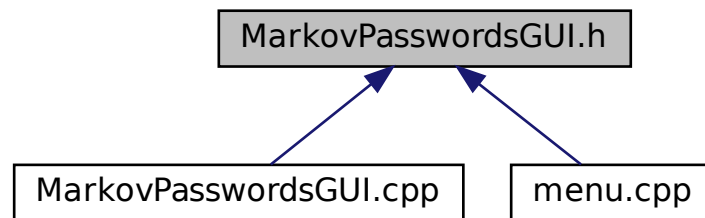
#include <QtWidgets/MainWindow>
#include "ui_MarkovPasswordsGUI.h"

```

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

9.34 MarkovPasswordsGUI.h

```

00001 #pragma once
00002
00003 #include <QtWidgets/QMainWindow>
00004 #include "ui_MarkovPasswordsGUI.h"
00005
00006
00007 namespace Markov::GUI{
00008     /** @brief Reporting UI.
00009     *
00010     * UI for reporting and debugging tools for MarkovPassword
00011     */
00012     class MarkovPasswordsGUI : public QMainWindow {
00013     Q_OBJECT
00014     public:
00015         /** @brief Default QT constructor.
00016         */
  
```

```

00017     * @param parent - Parent widget.
00018     */
00019     MarkovPasswordsGUI(QWidget *parent = Q_NULLPTR);
00020
00021     /** @brief Render a HTML file.
00022     * @param filename - Filename of the html file. (relative path to the views folder).
00023     */
00024     void renderHTMLFile(std::string* filename);
00025
00026     /** @brief Load a dataset to current view..
00027     * @param filename - Filename of the dataset file. (relative path to the views folder).
00028     */
00029     void loadDataset(std::string* filename);
00030
00031 private:
00032     Ui::MarkovPasswordsGUIClass ui;
00033
00034
00035     //Slots for buttons in GUI.
00036 public slots:
00037
00038     void MarkovPasswordsGUI::benchmarkSelected();
00039     void MarkovPasswordsGUI::modelvisSelected();
00040     void MarkovPasswordsGUI::visualDebugSelected();
00041     void MarkovPasswordsGUI::comparisonSelected();
00042 };
00043 };

```

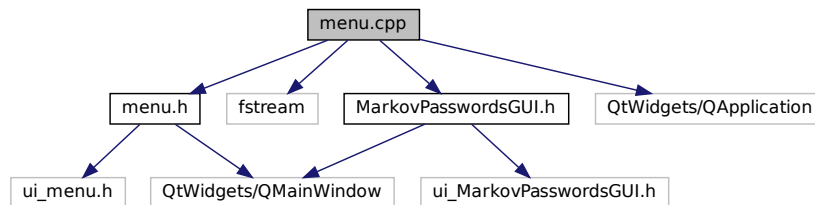
9.35 menu.cpp File Reference

```

#include "menu.h"
#include <fstream>
#include "MarkovPasswordsGUI.h"
#include <QtWidgets/QApplication>

```

Include dependency graph for menu.cpp:



9.36 menu.cpp

```

00001 #include "menu.h"
00002 #include <fstream>
00003 #include "MarkovPasswordsGUI.h"
00004 #include <QtWidgets/QApplication>
00005
00006 using namespace Markov::GUI;
00007
00008 menu::menu(QWidget* parent)
00009     : QMainWindow(parent)
00010 {
00011     ui.setupUi(this);
00012
00013
00014     //QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {about(); });
00015     QObject::connect(ui.visu, &QPushButton::clicked, this, [this] {visualization(); });
00016 }
00017 void menu::about() {
00018
00019
00020 }
00021 void menu::visualization() {
00022     MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
00023     w->show();
00024     this->close();

```

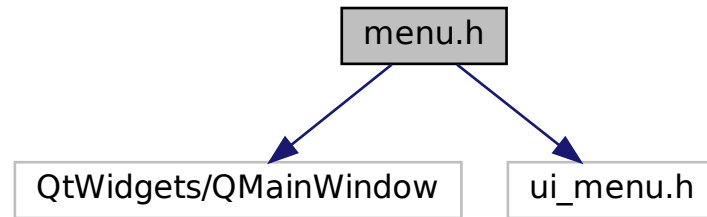
```
00025 }
```

9.37 menu.h File Reference

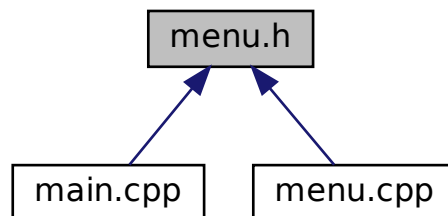
```
#include <QtWidgets/QMainWindow>
```

```
#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

9.38 menu.h

```
00001 #pragma once
00002 #include <QtWidgets/QMainWindow>
```

```

00003 #include "ui_menu.h"
00004
00005
00006 namespace Markov::GUI{
00007     /** @brief QT Menu class
00008     */
00009     class menu:public QMainWindow {
00010     Q_OBJECT
00011     public:
00012         menu(QWidget* parent = Q_NULLPTR);
00013
00014     private:
00015         Ui::main ui;
00016
00017     public slots:
00018         void about();
00019         void visualization();
00020     };
00021 };

```

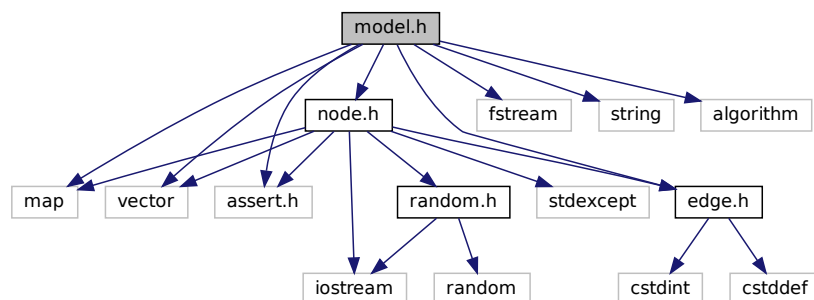
9.39 model.h File Reference

```

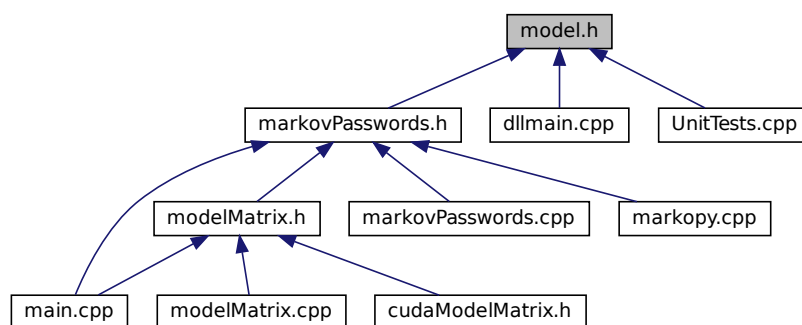
#include <map>
#include <vector>
#include <fstream>
#include <assert.h>
#include <string>
#include <algorithm>
#include "node.h"
#include "edge.h"

```

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.

9.40 model.h

```

00001 /** @dir Model.h
00002  *
00003  */
00004
00005
00006 #pragma once
00007 #include <map>
00008 #include <vector>
00009 #include <fstream>
00010 #include <assert.h>
00011 #include <string>
00012 #include <algorithm>
00013 #include "node.h"
00014 #include "edge.h"
00015
00016 /**
00017  @brief Namespace for the markov-model related classes.
00018  Contains Model, Node and Edge classes
00019  */
00020 namespace Markov {
00021
00022     template <typename NodeStorageType>
00023     class Node;
00024
00025     template <typename NodeStorageType>
00026     class Edge;
00027
00028     template <typename NodeStorageType>
00029
00030     /** @brief class for the final Markov Model, constructed from nodes and edges.
00031      *
00032      * Each atomic piece of the generation result is stored in a node, while edges contain the
00033      relation weights.
00034      * *Extending:*
00035      * To extend the class, implement the template and inherit from it, as "class MyModel : public
00036      Markov::Model<char>".
00037      * For a complete demonstration of how to extend the class, see MarkovPasswords.
00038      * Whole model can be defined as a list of the edges, as dangling nodes are pointless. This
00039      approach is used for the import/export operations.
00040      * For more information on importing/exporting model, check out the github readme and wiki page.
00041      */
00042     class Model {
00043     public:
00044
00045         /** @brief Initialize a model with only start and end nodes.
00046          *
00047          * Initialize an empty model with only a starterNode
00048          * Starter node is a special kind of node that has constant 0x00 value, and will be used to
00049          initiate the generation execution from.
00050          */
00051         Model<NodeStorageType>();
00052
00053         /** @brief Do a random walk on this model.
00054          *
00055          * Start from the starter node, on each node, invoke RandomNext using the random engine on
00056          current node, until terminator node is reached.
00057          * If terminator node is reached before minimum length criateria is reached, ignore the last
00058          selection and re-invoke randomNext
00059          */
00060     
```

```

00056      * If maximum length criteria is reached but final node is not, cut off the generation and
    proceed to the final node.
00057      * This function takes Markov::Random::RandomEngine as a parameter to generate pseudo random
    numbers from
00058      *
00059      * This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne
    output is higher in entropy, most use cases
00060      * don't really need super high entropy output, so Markov::Random::Marsaglia is preferable for
    better performance.
00061      *
00062      * This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening
    via maximum length criteria.
00063      *
00064      * @b Example @b Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use
    Marsaglia
00065      * @code{.cpp}
00066      * Markov::Model<char> model;
00067      * Model.import("model.mdl");
00068      * char* res = new char[11];
00069      * Markov::Random::Marsaglia MarsagliaRandomEngine;
00070      * for (int i = 0; i < 10; i++) {
00071      *     this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
00072      *     std::cout << res << "\n";
00073      * }
00074      * @endcode
00075      *
00076      * @param randomEngine Random Engine to use for the random walks. For examples, see
    Markov::Random::Mersenne and Markov::Random::Marsaglia
00077      * @param minSetting Minimum number of characters to generate
00078      * @param maxSetting Maximum number of character to generate
00079      * @param buffer buffer to write the result to
00080      * @return Null terminated string that was generated.
00081      */
00082      NodeStorageType* RandomWalk(Markov::Random::RandomEngine* randomEngine, int minSetting, int
    maxSetting, NodeStorageType* buffer);
00083
00084      /** @brief Adjust the model with a single string.
00085      *
00086      * Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from
    current node to the next, until NULL character is reached.
00087      *
00088      * Then, update the edge EdgeWeight from current node, to the terminator node.
00089      *
00090      * This function is used for training purposes, as it can be used for adjusting the model with
    each line of the corpus file.
00091      *
00092      * @b Example @b Use: Create an empty model and train it with string: "testdata"
00093      * @code{.cpp}
00094      * Markov::Model<char> model;
00095      * char test[] = "testdata";
00096      * model.AdjustEdge(test, 15);
00097      * @endcode
00098      *
00099      *
00100      * @param string - String that is passed from the training, and will be used to AdjustEdge the
    model with
00101      * @param occurrence - Occurrence of this string.
00102      *
00103      *
00104      */
00105      void AdjustEdge(const NodeStorageType* payload, long int occurrence);
00106
00107      /** @brief Import a file to construct the model.
00108      *
00109      * File contains a list of edges. For more info on the file format, check out the wiki and
    github readme pages.
00110      * Format is: Left_repr;EdgeWeight;right_repr
00111      *
00112      * Iterate over this list, and construct nodes and edges accordingly.
00113      * @return True if successful, False for incomplete models or corrupt file formats
00114      *
00115      * @b Example @b Use: Import a file from ifstream
00116      * @code{.cpp}
00117      * Markov::Model<char> model;
00118      * std::ifstream file("test.mdl");
00119      * model.Import(&file);
00120      * @endcode
00121      */
00122      bool Import(std::ifstream*);
00123
00124      /** @brief Open a file to import with filename, and call bool Model::Import with std::ifstream
00125      * @return True if successful, False for incomplete models or corrupt file formats
00126      *
00127      * @b Example @b Use: Import a file with filename
00128      * @code{.cpp}
00129      * Markov::Model<char> model;
00130      * model.Import("test.mdl");

```

```

00131         * @endcode
00132     */
00133     bool Import(const char* filename);
00134
00135     /** @brief Export a file of the model.
00136     *
00137     * File contains a list of edges.
00138     * Format is: Left_repr;EdgeWeight;right_repr.
00139     * For more information on the format, check out the project wiki or github readme.
00140     *
00141     * Iterate over this vertices, and their edges, and write them to file.
00142     * @return True if successful, False for incomplete models.
00143     *
00144     * @b Example @b Use: Export file to ofstream
00145     * @code{.cpp}
00146     * Markov::Model<char> model;
00147     * std::ofstream file("test.mdl");
00148     * model.Export(&file);
00149     * @endcode
00150     */
00151     bool Export(std::ofstream*);
00152
00153     /** @brief Open a file to export with filename, and call bool Model::Export with std::ofstream
00154     * @return True if successful, False for incomplete models or corrupt file formats
00155     *
00156     * @b Example @b Use: Export file to filename
00157     * @code{.cpp}
00158     * Markov::Model<char> model;
00159     * model.Export("test.mdl");
00160     * @endcode
00161     */
00162     bool Export(const char* filename);
00163
00164     /** @brief Return starter Node
00165     * @return starter node with 00 NodeValue
00166     */
00167     Node<NodeStorageType>* StarterNode() { return starterNode; }
00168
00169     /** @brief Return a vector of all the edges in the model
00170     * @return vector of edges
00171     */
00172     std::vector<Edge<NodeStorageType>*>* Edges() { return &edges; }
00173
00174     /** @brief Return starter Node
00175     * @return starter node with 00 NodeValue
00176     */
00177     std::map<NodeStorageType, Node<NodeStorageType>*>* Nodes() { return &nodes; }
00178
00179 private:
00180     /** @brief Map LeftNode is the Nodes NodeValue
00181     * Map RightNode is the node pointer
00182     */
00183     std::map<NodeStorageType, Node<NodeStorageType>*> nodes;
00184
00185     /** @brief Starter Node of this model.
00186     *
00187     */
00188     Node<NodeStorageType>* starterNode;
00189
00190
00191     /** @brief A list of all edges in this model.
00192     *
00193     */
00194     std::vector<Edge<NodeStorageType>*> edges;
00195 };
00196
00197 };
00198
00199 template <typename NodeStorageType>
00200 Markov::Model<NodeStorageType>::Model() {
00201     this->starterNode = new Markov::Node<NodeStorageType>(0);
00202     this->nodes.insert({ 0, this->starterNode });
00203 }
00204
00205 template <typename NodeStorageType>
00206 bool Markov::Model<NodeStorageType>::Import(std::ifstream* f) {
00207     std::string cell;
00208
00209     char src;
00210     char target;
00211     long int oc;
00212
00213     while (std::getline(*f, cell)) {
00214         //std::cout << "cell: " << cell << std::endl;
00215         src = cell[0];
00216         target = cell[cell.length() - 1];
00217         char* j;

```

```

00218         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00219         //std::cout << "oc << "\n";
00220         Markov::Node<NodeStorageType>* srcN;
00221         Markov::Node<NodeStorageType>* targetN;
00222         Markov::Edge<NodeStorageType>* e;
00223         if (this->nodes.find(src) == this->nodes.end()) {
00224             srcN = new Markov::Node<NodeStorageType>(src);
00225             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00226             //std::cout << "Creating new node at start.\n";
00227         }
00228         else {
00229             srcN = this->nodes.find(src)->second;
00230         }
00231
00232         if (this->nodes.find(target) == this->nodes.end()) {
00233             targetN = new Markov::Node<NodeStorageType>(target);
00234             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00235             //std::cout << "Creating new node at end.\n";
00236         }
00237         else {
00238             targetN = this->nodes.find(target)->second;
00239         }
00240         e = srcN->Link(targetN);
00241         e->AdjustEdge(oc);
00242         this->edges.push_back(e);
00243
00244         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--" <<
int(targetN->NodeValue()) << "\n";
00245
00246     }
00247
00248     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00249         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00250         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [] (Edge<NodeStorageType> *lhs,
00251 Edge<NodeStorageType> *rhs)->bool{
00252             return lhs->EdgeWeight() > rhs->EdgeWeight();
00253         });
00254         //for(int i=0; i<x.second->edgesV.size(); i++)
00255         // std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00256         //std::cout << "\n";
00257     }
00258     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00259     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00260
00261     return true;
00262 }
00263
00264 template <typename NodeStorageType>
00265 bool Markov::Model<NodeStorageType>::Import(const char* filename) {
00266     std::ifstream importfile;
00267     importfile.open(filename);
00268     return this->Import(&importfile);
00269 }
00270
00271 template <typename NodeStorageType>
00272 bool Markov::Model<NodeStorageType>::Export(std::ofstream* f) {
00273     Markov::Edge<NodeStorageType>* e;
00274     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00275         e = this->edges[i];
00276         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
e->RightNode()->NodeValue() << "\n";
00277         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
"\n";
00278     }
00279 }
00280
00281     return true;
00282 }
00283
00284 template <typename NodeStorageType>
00285 bool Markov::Model<NodeStorageType>::Export(const char* filename) {
00286     std::ofstream exportfile;
00287     exportfile.open(filename);
00288     return this->Export(&exportfile);
00289 }
00290
00291 template <typename NodeStorageType>
00292 NodeStorageType* Markov::Model<NodeStorageType>::RandomWalk(Markov::Random::RandomEngine*
randomEngine, int minSetting, int maxSetting, NodeStorageType* buffer) {
00293     Markov::Node<NodeStorageType>* n = this->starterNode;
00294     int len = 0;
00295     Markov::Node<NodeStorageType>* temp_node;
00296     while (true) {
00297         temp_node = n->RandomNext(randomEngine);
00298         if (len >= maxSetting) {
00299             break;

```

```

00300     }
00301     else if ((temp_node == NULL) && (len < minSetting)) {
00302         continue;
00303     }
00304
00305     else if (temp_node == NULL) {
00306         break;
00307     }
00308
00309     n = temp_node;
00310
00311     buffer[len++] = n->NodeValue();
00312 }
00313
00314 //null terminate the string
00315 buffer[len] = 0x00;
00316
00317 //do something with the generated string
00318 return buffer; //for now
00319 }
00320
00321 template <typename NodeStorageType>
00322 void Markov::Model<NodeStorageType>::AdjustEdge(const NodeStorageType* payload, long int occurrence) {
00323     NodeStorageType p = payload[0];
00324     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00325     Markov::Edge<NodeStorageType>* e;
00326     int i = 0;
00327
00328     if (p == 0) return;
00329     while (p != 0) {
00330         e = curnode->FindEdge(p);
00331         if (e == NULL) return;
00332         e->AdjustEdge(occurrence);
00333         curnode = e->RightNode();
00334         p = payload[++i];
00335     }
00336
00337     e = curnode->FindEdge('\xff');
00338     e->AdjustEdge(occurrence);
00339     return;
00340 }

```

9.41 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

9.42 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")

```

9.43 modelMatrix.cpp File Reference

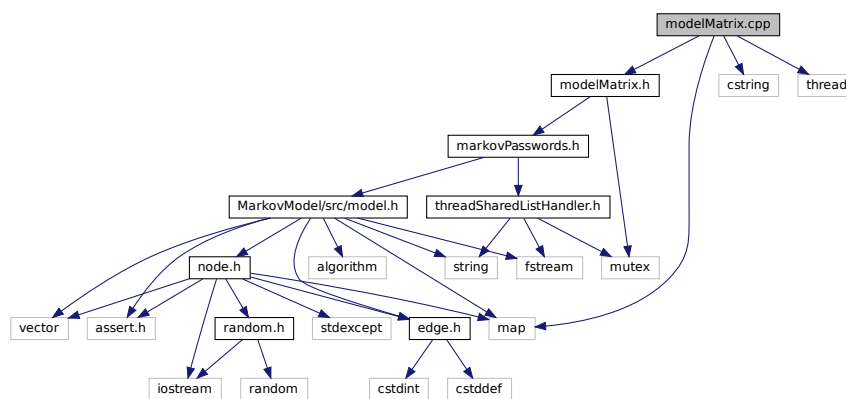
```
#include "modelMatrix.h"
```

```
#include <map>
```

```
#include <cstring>
```

```
#include <thread>
```

Include dependency graph for modelMatrix.cpp:



9.44 modelMatrix.cpp

```

00001 #include "modelMatrix.h"
00002 #include <map>
00003 #include <cstring>
00004 #include <thread>
00005
00006 Markov::API::ModelMatrix::ModelMatrix() {
00007
00008 }
00009
00010
00011 void Markov::API::ModelMatrix::ConstructMatrix() {
00012     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00013
00014     this->matrixIndex = new char[this->matrixSize];
00015     this->totalEdgeWeights = new long int[this->matrixSize];
00016
00017     this->edgeMatrix = new char*[this->matrixSize];
00018     for(int i=0; i<this->matrixSize; i++){
00019         this->edgeMatrix[i] = new char[this->matrixSize];
00020     }
00021     this->valueMatrix = new long int*[this->matrixSize];
00022     for(int i=0; i<this->matrixSize; i++){
00023         this->valueMatrix[i] = new long int[this->matrixSize];
00024     }
00025     std::map< char, Node< char > * > *nodes;
00026     nodes = this->Nodes();
00027     int i=0;
00028     for (auto const& [repr, node] : *nodes){
00029         if(repr!=0) this->matrixIndex[i] = repr;
00030         else this->matrixIndex[i] = 199;
00031         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00032         for(int j=0; j<this->matrixSize; j++){
00033             char val = node->NodeValue();
00034             if(val < 0) {

```

```

00035         for(int k=0;k<this->matrixSize;k++){
00036             this->valueMatrix[i][k] = 0;
00037             this->edgeMatrix[i][k] = 255;
00038         }
00039         break;
00040     }
00041     else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00042         this->valueMatrix[i][j] = 0;
00043         this->edgeMatrix[i][j] = 255;
00044     }else if(j==(this->matrixSize-1)) {
00045         this->valueMatrix[i][j] = 0;
00046         this->edgeMatrix[i][j] = 255;
00047     }else{
00048         this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00049         this->edgeMatrix[i][j] = node->edgesV[j]->RightNode()->NodeValue();
00050     }
00051 }
00052 }
00053 i++;
00054 }
00055 //this->DumpJSON();
00056 }
00057 }
00058
00059 void Markov::API::ModelMatrix::DumpJSON() {
00060     std::cout << "{\n  \"index\": \"\n";
00061     for(int i=0;i<this->matrixSize;i++){
00062         if(this->matrixIndex[i]==' ') std::cout << "\\ ";
00063         else if(this->matrixIndex[i]=='\\') std::cout << "\\\\";
00064         else if(this->matrixIndex[i]==0) std::cout << "\\x00";
00065         else if(i==0) std::cout << "\\xff";
00066         else if(this->matrixIndex[i]=='\n') std::cout << "\\n";
00067         else std::cout << this->matrixIndex[i];
00068     }
00069     std::cout << "\n";
00070     std::cout <<
00071     "\",\n"
00072     "\"  \"edgemap\": {\n";
00073     for(int i=0;i<this->matrixSize;i++){
00074         if(this->matrixIndex[i]==' ') std::cout << "    \"\\ \": [";
00075         else if(this->matrixIndex[i]=='\\') std::cout << "    \"\\\\ \": [";
00076         else if(this->matrixIndex[i]==0) std::cout << "    \"\\x00 \": [";
00077         else if(this->matrixIndex[i]<0) std::cout << "    \"\\xff \": [";
00078         else std::cout << "    \" \" < this->matrixIndex[i] < \" \": [";
00079         for(int j=0;j<this->matrixSize;j++){
00080             if(this->edgeMatrix[i][j]==' ') std::cout << "    \"\\ \": [";
00081             else if(this->edgeMatrix[i][j]=='\\') std::cout << "    \"\\\\ \": [";
00082             else if(this->edgeMatrix[i][j]==0) std::cout << "    \"\\x00 \": [";
00083             else if(this->edgeMatrix[i][j]<0) std::cout << "    \"\\xff \": [";
00084             else if(this->matrixIndex[i]=='\n') std::cout << "    \"\\n \": [";
00085             else std::cout << "    \" \" < this->edgeMatrix[i][j] < \" \": [";
00086             if(j!=this->matrixSize-1) std::cout << ", ";
00087         }
00088         std::cout << "],\n";
00089     }
00090     std::cout << "},\n";
00091     std::cout << "  \"weightmap\": {\n";
00092     for(int i=0;i<this->matrixSize;i++){
00093         if(this->matrixIndex[i]==' ') std::cout << "    \"\\ \": [";
00094         else if(this->matrixIndex[i]=='\\') std::cout << "    \"\\\\ \": [";
00095         else if(this->matrixIndex[i]==0) std::cout << "    \"\\x00 \": [";
00096         else if(this->matrixIndex[i]<0) std::cout << "    \"\\xff \": [";
00097         else std::cout << "    \" \" < this->matrixIndex[i] < \" \": [";
00098         for(int j=0;j<this->matrixSize;j++){
00099             std::cout << this->valueMatrix[i][j];
00100             if(j!=this->matrixSize-1) std::cout << ", ";
00101         }
00102         std::cout << "],\n";
00103     }
00104     std::cout << "  }\n\n";
00105 }
00106 }
00107 }
00108 }
00109 }
00110 }
00111 }
00112 void Markov::API::ModelMatrix::FastRandomWalkThread(std::mutex *mlock, std::ofstream *wordlist,
00113     unsigned long int n, int minLen, int maxLen, int id, bool bFileIO){
00114     if(n==0) return;
00115     Markov::Random::Marsaglia MarsagliaRandomEngine;
00116     char* e;
00117     char *res = new char[maxLen*n];
00118     int index = 0;
00119     char next;
00120     int len=0;

```

```

00121     long int selection;
00122     char cur;
00123     long int bufferctr = 0;
00124     for (int i = 0; i < n; i++) {
00125         cur=199;
00126         len=0;
00127         while (true) {
00128             e = strchr(this->matrixIndex, cur);
00129             index = e - this->matrixIndex;
00130             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00131             for(int j=0;j<this->matrixSize;j++){
00132                 selection -= this->valueMatrix[index][j];
00133                 if (selection < 0){
00134                     next = this->edgeMatrix[index][j];
00135                     break;
00136                 }
00137             }
00138
00139             if (len >= maxLen) break;
00140             else if ((next < 0) && (len < minLen)) continue;
00141             else if (next < 0) break;
00142             cur = next;
00143             res[bufferctr + len++] = cur;
00144         }
00145         res[bufferctr + len++] = '\n';
00146         bufferctr+=len;
00147     }
00148 }
00149 if(bFileIO){
00150     mlock->lock();
00151     *wordlist « res;
00152     mlock->unlock();
00153 }else{
00154     mlock->lock();
00155     std::cout « res;
00156     mlock->unlock();
00157 }
00158 delete res;
00159
00160 }
00161
00162
00163 void Markov::API::ModelMatrix::FastRandomWalk(unsigned long int n, const char* wordlistFileName, int
minLen, int maxLen, int threads, bool bFileIO){
00164
00165
00166     std::ofstream wordlist;
00167     if(bFileIO)
00168         wordlist.open(wordlistFileName);
00169
00170     std::mutex mlock;
00171     if(n<=50000000ull) return this->FastRandomWalkPartition(&mlock, &wordlist, n, minLen, maxLen,
bFileIO, threads);
00172     else{
00173         int numberOfPartitions = n/50000000ull;
00174         for(int i=0;i<numberOfPartitions;i++){
00175             this->FastRandomWalkPartition(&mlock, &wordlist, 50000000ull, minLen, maxLen, bFileIO,
threads);
00176         }
00177
00178
00179 }
00180
00181
00182 void Markov::API::ModelMatrix::FastRandomWalkPartition(std::mutex *mlock, std::ofstream *wordlist,
unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads){
00183
00184     int iterationsPerThread = n/threads;
00185     int iterationsPerThreadCarryOver = n%threads;
00186
00187     std::vector<std::thread*> threadsV;
00188
00189     int id = 0;
00190     for(int i=0;i<threads;i++){
00191         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00192         id++;
00193     }
00194
00195     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00196
00197     for(int i=0;i<threads;i++){
00198         threadsV[i]->join();
00199     }
00200 }

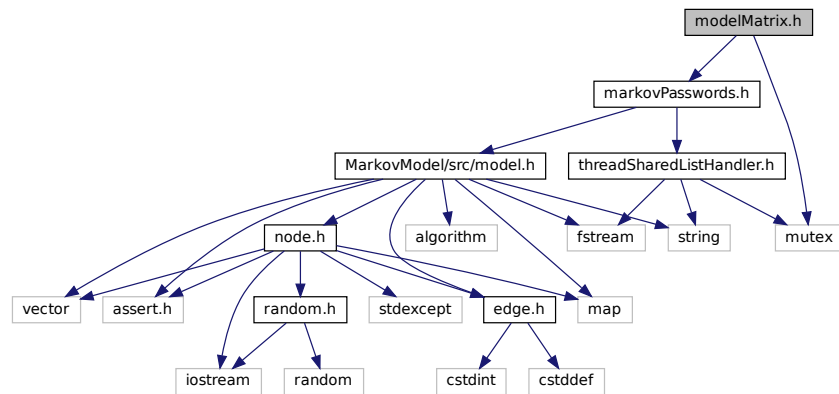
```


9.45 modelMatrix.h File Reference

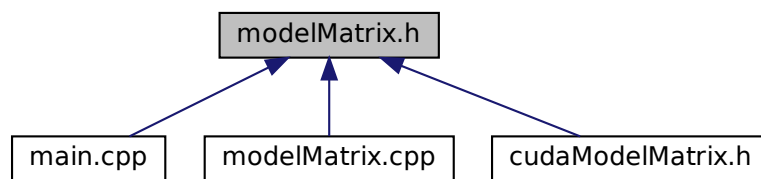
```
#include "markovPasswords.h"
```

```
#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.

9.46 modelMatrix.h

```

00001 #include "markovPasswords.h"
00002 #include <mutex>
00003
00004 namespace Markov::API{
00005
00006     /** @brief Class to flatten and reduce Markov::Model to a Matrix
00007     *

```

```

00008      * Matrix level operations can be used for Generation events, with a significant performance
00009      optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)
00010      * To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for
00011      allocation. Threads are synchronized and files are flushed every 50M operations.
00012      */
00013      class ModelMatrix : public Markov::API::MarkovPasswords{
00014      public:
00015          ModelMatrix();
00016
00017          /** @brief Construct the related Matrix data for the model.
00018          *
00019          * This operation can be used after importing/training to allocate and populate the matrix
00020          content.
00021          *
00022          * this will initialize:
00023          * char** edgeMatrix -> a 2D array of mapping left and right connections of each edge.
00024          * long int **valueMatrix -> a 2D array representing the edge weights.
00025          * int matrixSize -> Size of the matrix, aka total number of nodes.
00026          * char* matrixIndex -> order of nodes in the model
00027          * long int *totalEdgeWeights -> total edge weights of each Node.
00028          */
00029          void ConstructMatrix();
00030
00031          /** @brief Debug function to dump the model to a JSON file.
00032          *
00033          * Might not work 100%. Not meant for production use.
00034          */
00035          void DumpJSON();
00036
00037          /** @brief Random walk on the Matrix-reduced Markov::Model
00038          *
00039          * This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are
00040          partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
00041          *
00042          * If n>50M, threads are going to be synced, files are going to be flushed, and buffers will
00043          be reallocated every 50M generations.
00044          * This comes at a minor performance penalty.
00045          *
00046          * While it has the same functionality, this operation reduces
00047          Markov::API::MarkovPasswords::Generate runtime by %96.5
00048          *
00049          * This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually
00050          replace it.
00051          *
00052          * @param n - Number of passwords to generate.
00053          * @param wordlistFileName - Filename to write to
00054          * @param minLen - Minimum password length to generate
00055          * @param maxLen - Maximum password length to generate
00056          * @param threads - number of OS threads to spawn
00057          * @param bFileIO - If false, filename will be ignored and will output to stdout.
00058          *
00059          * @code{.cpp}
00060          * Markov::API::ModelMatrix mp;
00061          * mp.Import("models/finished.mdl");
00062          * mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
00063          * @endcode
00064          */
00065          void FastRandomWalk(unsigned long int n, const char* wordlistFileName, int minLen=6, int
00066          maxLen=12, int threads=20, bool bFileIO=true);
00067
00068          protected:
00069
00070          /** @brief A single partition of FastRandomWalk event
00071          *
00072          * Since FastRandomWalk has to allocate its output buffer before operation starts and writes
00073          data in chunks,
00074          *
00075          * large n parameters would lead to huge memory allocations.
00076          * @b Without @b Partitioning:
00077          * - 50M results 12 characters max -> 550 Mb Memory allocation
00078          *
00079          * - 5B results 12 characters max -> 55 Gb Memory allocation
00080          *
00081          * - 50B results 12 characters max -> 550GB Memory allocation
00082          *
00083          * Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.
00084          *
00085          * @param mlock - mutex lock to distribute to child threads
00086          * @param wordlist - Reference to the wordlist file to write to
00087          * @param n - Number of passwords to generate.
00088          * @param wordlistFileName - Filename to write to
00089          * @param minLen - Minimum password length to generate

```

```

00086      * @param maxLen - Maximum password length to generate
00087      * @param threads - number of OS threads to spawn
00088      * @param bFileIO - If false, filename will be ignored and will output to stdout.
00089      *
00090      */
00091      */
00092      void FastRandomWalkPartition(std::mutex *mlock, std::ofstream *wordlist, unsigned long int n,
int minLen, int maxLen, bool bFileIO, int threads);
00093
00094      /** @brief A single thread of a single partition of FastRandomWalk
00095      *
00096      * A FastRandomWalkPartition will initiate as many of this function as requested.
00097      *
00098      * This function contains the bulk of the generation algorithm.
00099      *
00100      * @param mlock - mutex lock to distribute to child threads
00101      * @param wordlist - Reference to the wordlist file to write to
00102      * @param n - Number of passwords to generate.
00103      * @param wordlistFileName - Filename to write to
00104      * @param minLen - Minimum password length to generate
00105      * @param maxLen - Maximum password length to generate
00106      * @param id - @b DEPRECATED Thread id - No longer used
00107      * @param bFileIO - If false, filename will be ignored and will output to stdout.
00108      *
00109      */
00110      */
00111      void FastRandomWalkThread(std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int
minLen, int maxLen, int id, bool bFileIO);
00112      char** edgeMatrix;
00113      long int **valueMatrix;
00114      int matrixSize;
00115      char* matrixIndex;
00116      long int *totalEdgeWeights;
00117      };
00118
00119
00120
00121 };

```

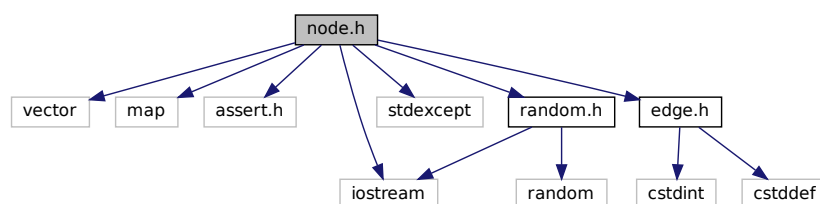
9.47 node.h File Reference

```

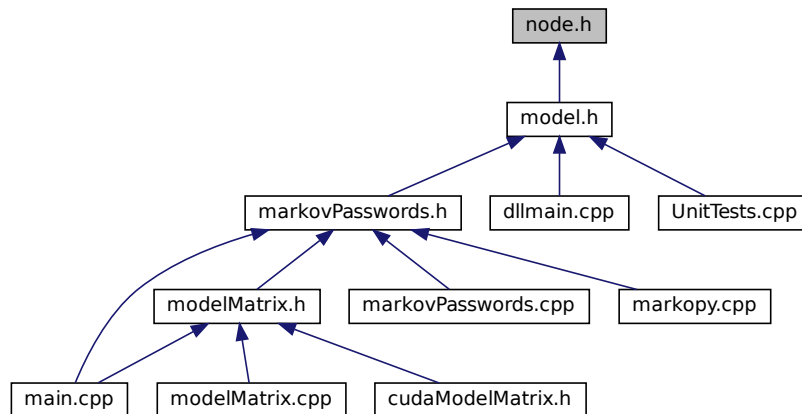
#include <vector>
#include <map>
#include <assert.h>
#include <iostream>
#include <stdexcept>
#include "edge.h"
#include "random.h"

```

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.

9.48 node.h

```

00001 #pragma once
00002 #include <vector>
00003 #include <map>
00004 #include <assert.h>
00005 #include <iostream>
00006 #include <stdexcept> // To use runtime_error
00007 #include "edge.h"
00008 #include "random.h"
00009 namespace Markov {
00010
00011     /** @brief A node class that for the vertices of model. Connected with eachother using Edge
00012     *
00013     * This class will later be templated to accept other data types than char*.
00014     */
00015     template <typename storageType>
00016     class Node {
00017     public:
00018
00019         /** @brief Default constructor. Creates an empty Node.
00020         */
00021         Node<storageType>();
00022
00023         /** @brief Constructor. Creates a Node with no edges and with given NodeValue.
00024         * @param _value - Nodes character representation.
00025         *
00026         * @b Example @b Use: Construct nodes
00027         * @code{.cpp}
00028         * Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00029         * Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00030         * @endcode
00031         */
00032         Node<storageType>(storageType _value);
00033
00034         /** @brief Link this node with another, with this node as its source.
00035         *
00036         * Creates a new Edge.
00037         */
00038     };
00039 
```

```

00037      * @param target - Target node which will be the RightNode() of new edge.
00038      * @return A new node with LeftNode as this, and RightNode as parameter target.
00039      *
00040      * @b Example @b Use: Construct nodes
00041      * @code{.cpp}
00042      * Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00043      * Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00044      * Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
00045      * @endcode
00046      */
00047      Edge<storageType>* Link(Node<storageType>*);
00048
00049      /** @brief Link this node with another, with this node as its source.
00050      *
00051      * *DOES NOT* create a new Edge.
00052      * @param Edge - Edge that will accept this node as its LeftNode.
00053      * @return the same edge as parameter target.
00054      *
00055      * @b Example @b Use: Construct and link nodes
00056      * @code{.cpp}
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 = LeftNode->Link(RightNode);
00060      * LeftNode->Link(e);
00061      * @endcode
00062      */
00063      Edge<storageType>* Link(Edge<storageType>*);
00064
00065      /** @brief Chose a random node from the list of edges, with regards to its EdgeWeight, and
00066      TraverseNode to that.
00067      *
00068      * This operation is done by generating a random number in range of 0-this.total_edge_weights,
00069      and then iterating over the list of edges.
00070      * At each step, EdgeWeight of the edge is subtracted from the random number, and once it is
00071      0, next node is selected.
00072      * @return Node that was chosen at EdgeWeight biased random.
00073      *
00074      * @b Example @b Use: Use randomNext to do a random walk on the model
00075      * @code{.cpp}
00076      * char* buffer[64];
00077      * Markov::Model<char> model;
00078      * model.Import("model.mdl");
00079      * Markov::Node<char>* n = model.starterNode;
00080      * int len = 0;
00081      * Markov::Node<char>* temp_node;
00082      * while (true) {
00083      *     temp_node = n->RandomNext(randomEngine);
00084      *     if (len >= maxSetting) {
00085      *         break;
00086      *     }
00087      *     else if ((temp_node == NULL) && (len < minSetting)) {
00088      *         continue;
00089      *     }
00090      *     else if (temp_node == NULL) {
00091      *         break;
00092      *     }
00093      *     n = temp_node;
00094      *     buffer[len++] = n->NodeValue();
00095      * }
00096      * @endcode
00097      */
00098      Node<storageType>* RandomNext(Markov::Random::RandomEngine* randomEngine);
00099
00100      /** @brief Insert a new edge to the this.edges.
00101      * @param edge - New edge that will be inserted.
00102      * @return true if insertion was successful, false if it fails.
00103      *
00104      * @b Example @b Use: Construct and update edges
00105      *
00106      * @code{.cpp}
00107      * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00108      * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00109      * Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00110      * Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00111      * Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00112      * e1->AdjustEdge(25);
00113      * src->UpdateEdges(e1);
00114      * e2->AdjustEdge(30);
00115      * src->UpdateEdges(e2);
00116      * @endcode
00117      */
00118      bool UpdateEdges(Edge<storageType>*);
00119
00120      /** @brief Find an edge with its character representation.

```

```

00121      * @param repr - character NodeValue of the target node.
00122      * @return Edge that is connected between this node, and the target node.
00123      *
00124      * @b Example @b Use: Construct and update edges
00125      *
00126      * @code{.cpp}
00127      * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00128      * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00129      * Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00130      * Markov::Edge<unsigned char>* res = NULL;
00131      * src->Link(target1);
00132      * src->Link(target2);
00133      * res = src->FindEdge('b');
00134      *
00135      * @endcode
00136      *
00137      */
00138      Edge<storageType>* FindEdge(storageType repr);
00139
00140      /** @brief Find an edge with its pointer. Avoid unless neccessary because comptutational cost
of find by character is cheaper (because of std::map)
00141      * @param target - target node.
00142      * @return Edge that is connected between this node, and the target node.
00143      */
00144      Edge<storageType>* FindEdge(Node<storageType>* target);
00145
00146      /** @brief Return character representation of this node.
00147      * @return character representation at _value.
00148      */
00149      inline unsigned char NodeValue();
00150
00151      /** @brief Change total weights with offset
00152      * @param offset to adjust the vertice weight with
00153      */
00154      void UpdateTotalVerticeWeight(long int offset);
00155
00156      /** @brief return edges
00157      */
00158      inline std::map<storageType, Edge<storageType>*>* Edges();
00159
00160      /** @brief return total edge weights
00161      */
00162      inline long int TotalEdgeWeights();
00163
00164
00165      std::vector<Edge<storageType>*> edgesV;
00166      private:
00167
00168
00169      storageType _value; /** @brief Character representation of this node. 0 for starter, 0xff for
terminator.*/
00170
00171      long int total_edge_weights; /** @brief Total weights of the vertices, required by
RandomNext;*/
00172
00173      /** @brief A map of all edges connected to this node, where this node is at the LeftNode.
00174      *
00175      * Map is indexed by unsigned char, which is the character representation of the node.
00176      */
00177      std::map<storageType, Edge<storageType>*> edges;
00178      };
00179 };
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189 template <typename storageType>
00190 Markov::Node<storageType>::Node(storageType _value) {
00191     this->_value = _value;
00192     this->total_edge_weights = 0L;
00193 };
00194
00195 template <typename storageType>
00196 Markov::Node<storageType>::Node() {
00197     this->_value = 0;
00198     this->total_edge_weights = 0L;
00199 };
00200
00201 template <typename storageType>
00202 inline unsigned char Markov::Node<storageType>::NodeValue() {
00203     return _value;
00204 }

```

```

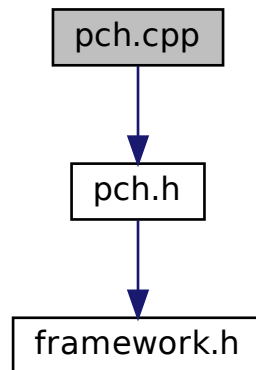
00205
00206 template <typename storageType>
00207 Markov::Edge<storageType>* Markov::Node<storageType>::Link(Markov::Node<storageType>* n) {
00208     Markov::Edge<storageType>* v = new Markov::Edge<storageType>(this, n);
00209     this->UpdateEdges(v);
00210     return v;
00211 }
00212
00213 template <typename storageType>
00214 Markov::Edge<storageType>* Markov::Node<storageType>::Link(Markov::Edge<storageType>* v) {
00215     v->SetLeftEdge(this);
00216     this->UpdateEdges(v);
00217     return v;
00218 }
00219
00220 template <typename storageType>
00221 Markov::Node<storageType>* Markov::Node<storageType>::RandomNext(Markov::Random::RandomEngine*
    randomEngine) {
00222
00223     //get a random NodeValue in range of total_vertice_weight
00224     long int selection = randomEngine->random() %
    this->total_edge_weights; //distribution() (generator()); // distribution(generator);
00225     //make absolute, no negative modulus values wanted
00226     //selection = (selection >= 0) ? selection : (selection + this->total_edge_weights);
00227     for(int i=0; i<this->edgesV.size(); i++){
00228         selection -= this->edgesV[i]->EdgeWeight();
00229         if (selection < 0) return this->edgesV[i]->TraverseNode();
00230     }
00231
00232     //if this assertion is reached, it means there is an implementation error above
00233     std::cout << "This should never be reached (node failed to walk to next)\n"; //cant assert from
    child thread
00234     assert(true && "This should never be reached (node failed to walk to next)");
00235     return NULL;
00236 }
00237
00238 template <typename storageType>
00239 bool Markov::Node<storageType>::UpdateEdges(Markov::Edge<storageType>* v) {
00240     this->edges.insert({ v->RightNode()->NodeValue(), v });
00241     this->edgesV.push_back(v);
00242     //this->total_edge_weights += v->EdgeWeight();
00243     return v->TraverseNode();
00244 }
00245
00246 template <typename storageType>
00247 Markov::Edge<storageType>* Markov::Node<storageType>::FindEdge(storageType repr) {
00248     auto e = this->edges.find(repr);
00249     if (e == this->edges.end()) return NULL;
00250     return e->second;
00251 };
00252
00253 template <typename storageType>
00254 void Markov::Node<storageType>::UpdateTotalVerticeWeight(long int offset) {
00255     this->total_edge_weights += offset;
00256 }
00257
00258 template <typename storageType>
00259 inline std::map<storageType, Markov::Edge<storageType>*>* Markov::Node<storageType>::Edges() {
00260     return &(this->edges);
00261 }
00262
00263 template <typename storageType>
00264 inline long int Markov::Node<storageType>::TotalEdgeWeights() {
00265     return this->total_edge_weights;
00266 }

```

9.49 pch.cpp File Reference

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

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



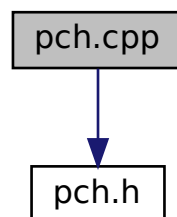
9.50 MarkovModel/src/pch.cpp

```
00001 // pch.cpp: source file corresponding to the pre-compiled header
00002
00003 #include "pch.h"
00004
00005 // When you are using pre-compiled headers, this source file is necessary for compilation to succeed.
```

9.51 pch.cpp File Reference

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

Include dependency graph for UnitTests/pch.cpp:



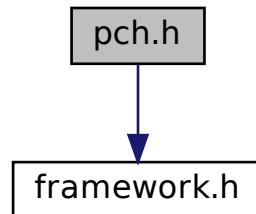
9.52 UnitTests/pch.cpp

```
00001 // pch.cpp: source file corresponding to the pre-compiled header
00002
00003 #include "pch.h"
00004
00005 // When you are using pre-compiled headers, this source file is necessary for compilation to succeed.
```

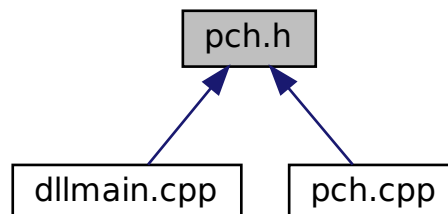

9.53 pch.h File Reference

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

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



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

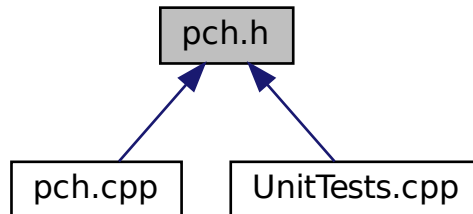


9.54 MarkovModel/src/pch.h

```
00001 // pch.h: This is a precompiled header file.
00002 // Files listed below are compiled only once, improving build performance for future builds.
00003 // This also affects IntelliSense performance, including code completion and many code browsing
00004 // features.
00004 // However, files listed here are ALL re-compiled if any one of them is updated between builds.
00005 // Do not add files here that you will be updating frequently as this negates the performance
00006 // advantage.
00006
00007 #ifndef PCH_H
00008 #define PCH_H
00009
00010 // add headers that you want to pre-compile here
00011 #include "framework.h"
00012
00013 #endif //PCH_H
```

9.55 pch.h File Reference

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



9.56 UnitTests/pch.h

```

00001 // pch.h: This is a precompiled header file.
00002 // Files listed below are compiled only once, improving build performance for future builds.
00003 // This also affects IntelliSense performance, including code completion and many code browsing
00004 // features.
00004 // However, files listed here are ALL re-compiled if any one of them is updated between builds.
00005 // Do not add files here that you will be updating frequently as this negates the performance
00005 // advantage.
00006
00007 #ifndef PCH_H
00008 #define PCH_H
00009
00010 // add headers that you want to pre-compile here
00011
00012 #endif //PCH_H
  
```

9.57 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

9.58 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")

```

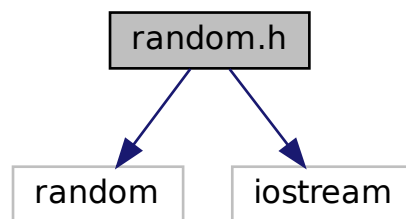
9.59 random.h File Reference

```

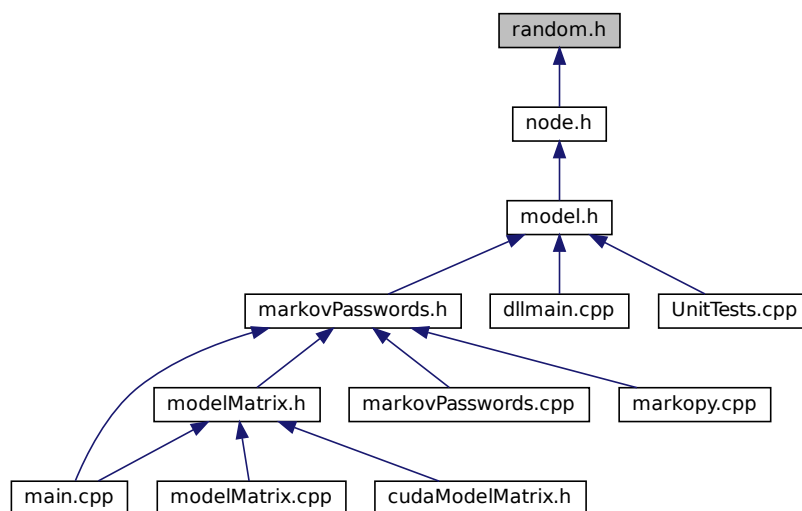
#include <random>
#include <iostream>

```

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.

9.60 random.h

```

00001
00002
00003 #include <random>
00004 #include <iostream>
00005
00006 /**
00007  * @brief Objects related to RNG
00008  */
00009 namespace Markov::Random{
00010
00011     /** @brief An abstract class for Random Engine
00012      *
00013      * This class is used for generating random numbers, which are used for random walking on the
00014      * graph.
00015      * Main reason behind allowing different random engines is that some use cases may favor
00016      * performance,
00017      * while some favor good random.
00018      * Mersenne can be used for truer random, while Marsaglia can be used for deterministic but fast
00019      * random.
00020      */
00021     class RandomEngine{
00022     public:
00023         virtual inline unsigned long random() = 0;
00024     };
00025
00026
00027
00028     /** @brief Implementation using Random.h default random engine
00029      *
00030      * This engine is also used by other engines for seeding.
00031      *
00032      * @b Example @b Use: Using Default Engine with RandomWalk
00033      * @code{.cpp}
00034      * Markov::Model<char> model;
00035      * Model.import("model.mdl");
00036      * char* res = new char[11];
00037      * Markov::Random::DefaultRandomEngine randomEngine;
00038      * for (int i = 0; i < 10; i++) {
00039      *     this->RandomWalk(&randomEngine, 5, 10, res);
00040      *     std::cout << res << "\n";
00041      * }
00042      * @endcode
00043      *
00044      * @b Example @b Use: Generating a random number with Marsaglia Engine
00045      * @code{.cpp}
00046      * Markov::Random::DefaultRandomEngine de;
00047      * std::cout << de.random();
00048      * @endcode
00049      *
00050      */
00051
00052     class DefaultRandomEngine : public RandomEngine{

```

```

00053     public:
00054         /** @brief Generate Random Number
00055          * @return random number in long range.
00056          */
00057         inline unsigned long random() {
00058             return this->distribution() (this->generator());
00059         }
00060     protected:
00061
00062         /** @brief Default random device for seeding
00063          *
00064          */
00065         inline std::random_device& rd() {
00066             static std::random_device _rd;
00067             return _rd;
00068         }
00069
00070         /** @brief Default random engine for seeding
00071          *
00072          */
00073         inline std::default_random_engine& generator() {
00074             static std::default_random_engine _generator(rd());
00075             return _generator;
00076         }
00077
00078         /** @brief Distribution schema for seeding.
00079          *
00080          */
00081         inline std::uniform_int_distribution<long long unsigned>& distribution() {
00082             static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffffff);
00083             return _distribution;
00084         }
00085     };
00086
00087
00088     /** @brief Implementation of Marsaglia Random Engine
00089     *
00090     * This is an implementation of Marsaglia Random engine, which for most use cases is a better fit
00091     than other solutions.
00092     * Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.
00093     *
00094     * This implementation of the Marsaglia Engine is seeded by random.h default random engine.
00095     * RandomEngine is only seeded once so its not a performance issue.
00096     *
00097     * @b Example @b Use: Using Marsaglia Engine with RandomWalk
00098     * @code{.cpp}
00099     * Markov::Model<char> model;
00100     * Model.import("model.mdl");
00101     * char* res = new char[11];
00102     * Markov::Random::Marsaglia MarsagliaRandomEngine;
00103     * for (int i = 0; i < 10; i++) {
00104     *     this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
00105     *     std::cout << res << "\n";
00106     * }
00107     * @endcode
00108     *
00109     * @b Example @b Use: Generating a random number with Marsaglia Engine
00110     * @code{.cpp}
00111     * Markov::Random::Marsaglia me;
00112     * std::cout << me.random();
00113     * @endcode
00114     *
00115     */
00116     class Marsaglia : public DefaultRandomEngine{
00117     public:
00118
00119         /** @brief Construct Marsaglia Engine
00120          *
00121          * Initialize x,y and z using the default random engine.
00122          */
00123         Marsaglia() {
00124             this->x = this->distribution() (this->generator());
00125             this->y = this->distribution() (this->generator());
00126             this->z = this->distribution() (this->generator());
00127             //std::cout << "x: " << x << ", y: " << y << ", z: " << z << "\n";
00128         }
00129
00130
00131         inline unsigned long random() {
00132             unsigned long t;
00133             x ^= x << 16;
00134             x ^= x >> 5;
00135             x ^= x << 1;
00136
00137             t = x;
00138             x = y;

```

```

00139         y = z;
00140         z = t ^ x ^ y;
00141
00142         return z;
00143     }
00144
00145     unsigned long x;
00146     unsigned long y;
00147     unsigned long z;
00148 };
00149
00150
00151
00152     /** @brief Implementation of Mersenne Twister Engine
00153     *
00154     * This is an implementation of Mersenne Twister Engine, which is slow but is a good
00155     * implementation for high entropy pseudorandom.
00156     *
00157     * @b Example @b Use: Using Mersenne Engine with RandomWalk
00158     * @code{.cpp}
00159     * Markov::Model<char> model;
00160     * Model.import("model.mdl");
00161     * char* res = new char[11];
00162     * Markov::Random::Mersenne MersenneTwisterEngine;
00163     * for (int i = 0; i < 10; i++) {
00164     *     this->RandomWalk(&MersenneTwisterEngine, 5, 10, res);
00165     *     std::cout << res << "\n";
00166     * }
00167     * @endcode
00168     *
00169     * @b Example @b Use: Generating a random number with Marsaglia Engine
00170     * @code{.cpp}
00171     * Markov::Random::Mersenne me;
00172     * std::cout << me.random();
00173     * @endcode
00174     *
00175     */
00176     class Mersenne : public DefaultRandomEngine{
00177     };
00178 };
00179
00180
00181 };

```

9.61 README.md File Reference

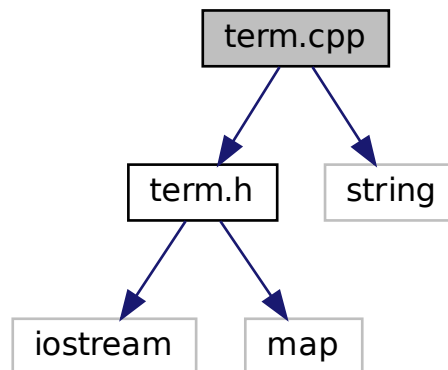
9.62 term.cpp File Reference

```

#include "term.h"
#include <string>

```

Include dependency graph for term.cpp:



Functions

- `std::ostream & operator<< (std::ostream &os, const Terminal::color &c)`

9.62.1 Function Documentation

9.62.1.1 operator<<()

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

overload for `std::cout`.

Definition at line 60 of file [term.cpp](#).

```
00060                                                                 {
00061     char buf[6];
00062     sprintf(buf, "%d", Terminal::colormap.find\(c\)->second);
00063     os << "\e[1;" << buf << "m";
00064     return os;
00065 }
```

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

9.63 term.cpp

```
00001 #pragma once
00002 #include "term.h"
00003 #include <string>
00004
00005 using namespace Markov::API::CLI;
00006
00007 //Windows text processing is different from unix systems, so use windows header and text attributes
00008 #ifdef _WIN32
00009
00010 HANDLE Terminal::\_stdout;
00011 HANDLE Terminal::\_stderr;
00012
00013 std::map<Terminal::color, DWORD> Terminal::colormap = {
00014     {Terminal::color::BLACK, 0},
00015     {Terminal::color::BLUE, 1},
00016     {Terminal::color::GREEN, 2},
00017     {Terminal::color::CYAN, 3},
00018     {Terminal::color::RED, 4},
00019     {Terminal::color::MAGENTA, 5},
```

```

00020     {Terminal::color::BROWN, 6},
00021     {Terminal::color::LIGHTGRAY, 7},
00022     {Terminal::color::DARKGRAY, 8},
00023     {Terminal::color::YELLOW, 14},
00024     {Terminal::color::WHITE, 15},
00025     {Terminal::color::RESET, 15},
00026 };
00027
00028
00029 Terminal::Terminal() {
00030     Terminal::_stdout = GetStdHandle(STD_OUTPUT_HANDLE);
00031     Terminal::_stderr = GetStdHandle(STD_ERROR_HANDLE);
00032 }
00033
00034 std::ostream& operator<<(std::ostream& os, const Terminal::color& c) {
00035     SetConsoleTextAttribute(Terminal::_stdout, Terminal::colormap.find(c)->second);
00036     return os;
00037 }
00038
00039 #else
00040
00041 std::map<Terminal::color, int> Terminal::colormap = {
00042     {Terminal::color::BLACK, 30},
00043     {Terminal::color::BLUE, 34},
00044     {Terminal::color::GREEN, 32},
00045     {Terminal::color::CYAN, 36},
00046     {Terminal::color::RED, 31},
00047     {Terminal::color::MAGENTA, 35},
00048     {Terminal::color::BROWN, 0},
00049     {Terminal::color::LIGHTGRAY, 0},
00050     {Terminal::color::DARKGRAY, 0},
00051     {Terminal::color::YELLOW, 33},
00052     {Terminal::color::WHITE, 37},
00053     {Terminal::color::RESET, 0},
00054 };
00055
00056 Terminal::Terminal() {
00057     /*this->*/
00058 }
00059
00060 std::ostream& operator<<(std::ostream& os, const Terminal::color& c) {
00061     char buf[6];
00062     sprintf(buf, "%d", Terminal::colormap.find(c)->second);
00063     os << "\e[" << buf << "m";
00064     return os;
00065 }
00066
00067
00068
00069
00070 #endif

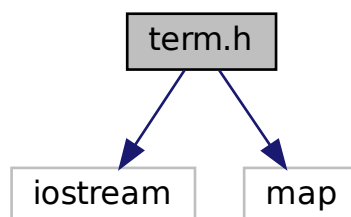
```

9.64 term.h File Reference

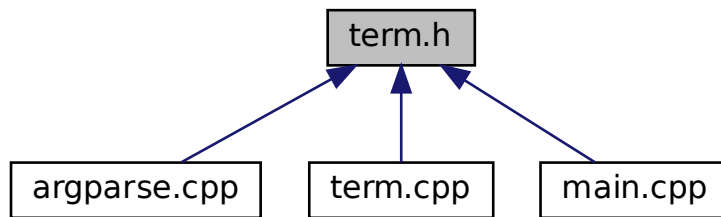
```
#include <iostream>
```

```
#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 arguments.

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)`

9.64.1 Macro Definition Documentation

9.64.1.1 TERM_FAIL

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

Definition at line 10 of file [term.h](#).

9.64.1.2 TERM_INFO

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

Definition at line 11 of file [term.h](#).

9.64.1.3 TERM_SUCC

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

Definition at line 13 of file [term.h](#).

9.64.1.4 TERM_WARN

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

Definition at line 12 of file [term.h](#).

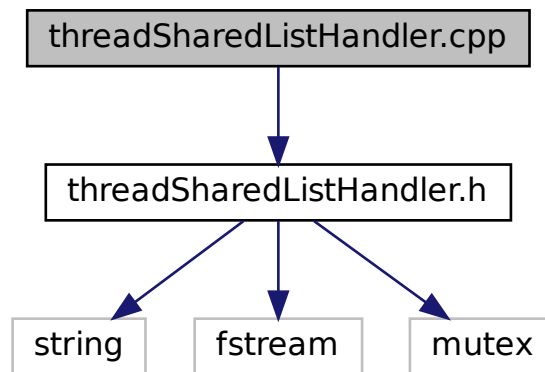
9.65 term.h

```
00001 #pragma once
00002
00003 #ifdef _WIN32
00004 #include <Windows.h>
00005 #endif
00006
00007 #include <iostream>
00008 #include <map>
00009
00010 #define TERM_FAIL "[" << Markov::API::CLI::Terminal::color::RED << "+" <<
00011 Markov::API::CLI::Terminal::color::RESET << "]" "
00012 #define TERM_INFO "[" << Markov::API::CLI::Terminal::color::BLUE << "+" <<
00013 Markov::API::CLI::Terminal::color::RESET << "]" "
00014 #define TERM_WARN "[" << Markov::API::CLI::Terminal::color::YELLOW << "+" <<
00015 Markov::API::CLI::Terminal::color::RESET << "]" "
00016 #define TERM_SUCC "[" << Markov::API::CLI::Terminal::color::GREEN << "+" <<
00017 Markov::API::CLI::Terminal::color::RESET << "]" "
00018
00019 namespace Markov::API::CLI{
00020     /** @brief pretty colors for Terminal. Windows Only.
00021     */
00022     class Terminal {
00023     public:
00024
00025         /** Default constructor.
00026         * Get references to stdout and stderr handles.
00027         */
00028         Terminal();
00029
00030         enum color { RESET, BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, LIGHTGRAY,
00031 DARKGRAY, BROWN };
00032
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         static std::ostream endl;
00042
00043     };
00044
00045     /** overload for std::cout.
00046     */
00047     std::ostream& operator<<(std::ostream& os, const Markov::API::CLI::Terminal::color& c);
00048
00049 }
```

9.66 threadSharedListHandler.cpp File Reference

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

Include dependency graph for threadSharedListHandler.cpp:



9.67 threadSharedListHandler.cpp

```

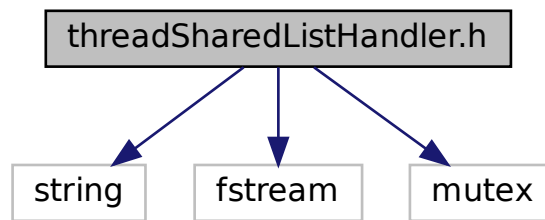
00001 #include "threadSharedListHandler.h"
00002
00003
00004 Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler(const char* filename){
00005     this->listfile;
00006     this->listfile.open(filename, std::ios_base::binary);
00007 }
00008
00009
00010 bool Markov::API::Concurrency::ThreadSharedListHandler::next(std::string* line){
00011     bool res = false;
00012     this->mlock.lock();
00013     res = (std::getline(this->listfile, *line, '\n')) ? true : false;
00014     this->mlock.unlock();
00015
00016     return res;
00017 }
  
```

9.68 threadSharedListHandler.h File Reference

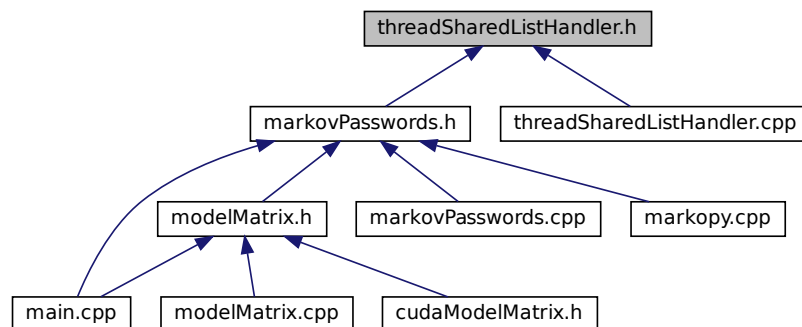
```

#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.

9.69 threadSharedListHandler.h

```

00001 #include <string>
00002 #include <fstream>
00003 #include <mutex>
00004
00005 /** @brief Namespace for Concurrency related classes
00006 */
00007 namespace Markov::API::Concurrency{
  
```

```

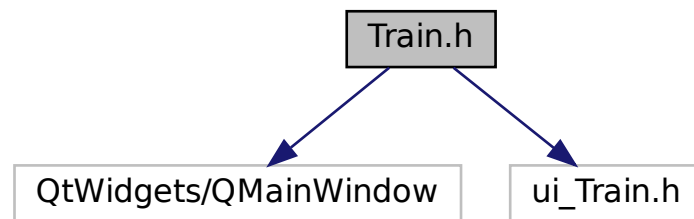
00008
00009 /** @brief Simple class for managing shared access to file
00010 *
00011 * This class maintains the handover of each line from a file to multiple threads.
00012 *
00013 * When two different threads try to read from the same file while reading a line isn't completed, it
00014 * can have unexpected results.
00015 * Line might be split, or might be read twice.
00016 * This class locks the read action on the list until a line is completed, and then proceeds with the
00017 * handover.
00018 */
00019 class ThreadSharedListHandler{
00020 public:
00021     /** @brief Construct the Thread Handler with a filename
00022     *
00023     * Simply open the file, and initialize the locks.
00024     *
00025     * @b Example @b Use: Simple file read
00026     * @code{.cpp}
00027     * ThreadSharedListHandler listhandler("test.txt");
00028     * std::string line;
00029     * std::cout << listhandler->next(&line) << "\n";
00030     * @endcode
00031     *
00032     * @b Example @b Use: Example use case from MarkovPasswords showing multithreaded access
00033     * @code{.cpp}
00034     * void MarkovPasswords::Train(const char* datasetFileName, char delimiter, int threads) {
00035     *     ThreadSharedListHandler listhandler(datasetFileName);
00036     *     auto start = std::chrono::high_resolution_clock::now();
00037     *     std::vector<std::thread*> threadsV;
00038     *     for(int i=0;i<threads;i++){
00039     *         threadsV.push_back(new std::thread(&MarkovPasswords::TrainThread, this, &listhandler,
00040     * datasetFileName, delimiter));
00041     *     }
00042     *     for(int i=0;i<threads;i++){
00043     *         threadsV[i]->join();
00044     *         delete threadsV[i];
00045     *     }
00046     *     auto finish = std::chrono::high_resolution_clock::now();
00047     *     std::chrono::duration<double> elapsed = finish - start;
00048     *     std::cout << "Elapsed time: " << elapsed.count() << " s\n";
00049     * }
00050     *
00051     * void MarkovPasswords::TrainThread(ThreadSharedListHandler *listhandler, const char*
00052     * datasetFileName, char delimiter){
00053     *     char format_str[] = "%ld,%s";
00054     *     format_str[2]=delimiter;
00055     *     std::string line;
00056     *     while (listhandler->next(&line)) {
00057     *         long int oc;
00058     *         if (line.size() > 100) {
00059     *             line = line.substr(0, 100);
00060     *         }
00061     *         char* linebuf = new char[line.length()+5];
00062     *         sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
00063     *         this->AdjustEdge((const char*)linebuf, oc);
00064     *         delete linebuf;
00065     *     }
00066     * }
00067     * @endcode
00068     *
00069     * @param filename Filename for the file to manage.
00070     */
00071     ThreadSharedListHandler(const char* filename);
00072
00073     /** @brief Read the next line from the file.
00074     *
00075     * This action will be blocked until another thread (if any) completes the read operation on the
00076     * file.
00077     *
00078     * @b Example @b Use: Simple file read
00079     * @code{.cpp}
00080     * ThreadSharedListHandler listhandler("test.txt");
00081     * std::string line;
00082     * std::cout << listhandler->next(&line) << "\n";
00083     * @endcode
00084     */
00085     bool next(std::string* line);
00086
00087 private:
00088     std::ifstream listfile;
00089     std::mutex mlock;

```

```
00090 };
00091
00092 };
```

9.70 Train.h File Reference

```
#include <QtWidgets/QMainWindow>
#include "ui_Train.h"
Include dependency graph for Train.h:
```



Classes

- class [Markov::GUI::Train](#)
QT Training page class.

Namespaces

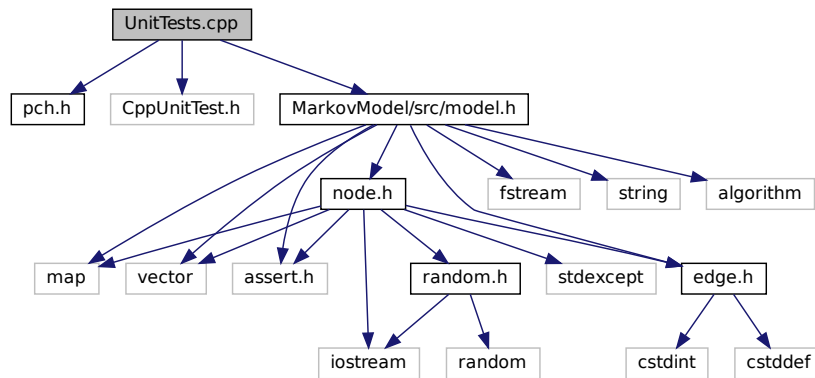
- [Markov](#)
Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.
- [Markov::GUI](#)
namespace for MarkovPasswords [API GUI](#) wrapper

9.71 Train.h

```
00001 #pragma once
00002 #include <QtWidgets/QMainWindow>
00003 #include "ui_Train.h"
00004
00005 namespace Markov::GUI{
00006
00007     /** @brief QT Training page class
00008     */
00009     class Train :public QMainWindow {
00010     Q_OBJECT
00011     public:
00012         Train(QWidget* parent = Q_NULLPTR);
00013
00014     private:
00015         Ui::Train ui;
00016
00017     public slots:
00018         void home();
00019         void train();
00020     };
00021 };
```

9.72 UnitTests.cpp File Reference

```
#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.

9.73 UnitTests.cpp

```

00001 #include "pch.h"
00002 #include "CppUnitTest.h"
00003 #include "MarkovModel/src/model.h"
00004
00005 using namespace Microsoft::VisualStudio::CppUnitTestFramework;
00006
00007
00008 /** @brief Namespace for Microsoft Native Unit Testing Classes
00009 */
00010 namespace Testing {
00011
00012     /** @brief Testing Namespace for Minimal Viable Product
00013     */
00014     namespace MVP {
00015         /** @brief Testing Namespace for MVP MarkovModel
00016         */
00017         namespace MarkovModel
00018         {
00019             /** @brief Test class for minimal viable Edge
00020             */
00021             TEST_CLASS(Edge)
00022             {
00023             public:
00024
00025                 /** @brief test default constructor
00026                 */
00027                 TEST_METHOD(default_constructor) {
00028                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>;
00029                     Assert::IsNull(e->LeftNode());
00030                     Assert::IsNull(e->RightNode());
00031                     delete e;
00032                 }
00033
00034                 /** @brief test linked constructor with two nodes
00035                 */
00036                 TEST_METHOD(linked_constructor) {
00037                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00038                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00039                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00040 RightNode);
00041                     Assert::IsTrue(LeftNode == e->LeftNode());
00042                     Assert::IsTrue(RightNode == e->RightNode());
00043                     delete LeftNode;
00044                     delete RightNode;
00045                     delete e;
00046                 }
00047
00048                 /** @brief test AdjustEdge function
00049                 */
00050                 TEST_METHOD(AdjustEdge) {
00051                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00052                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00053                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00054 RightNode);
00055                     e->AdjustEdge(15);
00056                     Assert::AreEqual(15ull, e->EdgeWeight());
00057                     e->AdjustEdge(15);
00058                     Assert::AreEqual(30ull, e->EdgeWeight());
00059                     delete LeftNode;
00060                     delete RightNode;
00061                     delete e;
00062                 }
00063
00064                 /** @brief test TraverseNode returning RightNode
00065                 */
00066                 TEST_METHOD(TraverseNode) {
00067                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00068                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00069                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00070 RightNode);
00071                     Assert::IsTrue(RightNode == e->TraverseNode());
00072                     delete LeftNode;
00073                     delete RightNode;
00074                     delete e;
00075                 }
00076
00077                 /** @brief test LeftNode/RightNode setter
00078                 */
00079                 TEST_METHOD(set_left_and_right) {
00080                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00081                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00082                     Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(LeftNode,
00083 RightNode);
00084                     Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>;

```



```

00082         e2->SetLeftEdge(LeftNode);
00083         e2->SetRightEdge(RightNode);
00084
00085         Assert::IsTrue(e1->LeftNode() == e2->LeftNode());
00086         Assert::IsTrue(e1->RightNode() == e2->RightNode());
00087         delete LeftNode;
00088         delete RightNode;
00089         delete e1;
00090         delete e2;
00091     }
00092
00093     /** @brief test negative adjustments
00094     */
00095     TEST_METHOD(negative_adjust) {
00096         Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00097         Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00098         Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
RightNode);
00099         e->AdjustEdge(15);
00100         Assert::AreEqual(15ull, e->EdgeWeight());
00101         e->AdjustEdge(-15);
00102         Assert::AreEqual(0ull, e->EdgeWeight());
00103         delete LeftNode;
00104         delete RightNode;
00105         delete e;
00106     }
00107 };
00108
00109     /** @brief Test class for minimal viable Node
00110     */
00111     TEST_CLASS(Node)
00112     {
00113     public:
00114
00115         /** @brief test default constructor
00116         */
00117         TEST_METHOD(default_constructor) {
00118             Markov::Node<unsigned char>* n = new Markov::Node<unsigned char>();
00119             Assert::AreEqual((unsigned char)0, n->NodeValue());
00120             delete n;
00121         }
00122
00123         /** @brief test custom constructor with unsigned char
00124         */
00125         TEST_METHOD(uchar_constructor) {
00126             Markov::Node<unsigned char>* n = NULL;
00127             unsigned char test_cases[] = { 'c', 0x00, 0xff, -32 };
00128             for (unsigned char tcase : test_cases) {
00129                 n = new Markov::Node<unsigned char>(tcase);
00130                 Assert::AreEqual(tcase, n->NodeValue());
00131                 delete n;
00132             }
00133         }
00134
00135         /** @brief test link function
00136         */
00137         TEST_METHOD(link_left) {
00138             Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00139             Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00140
00141             Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
00142             delete LeftNode;
00143             delete RightNode;
00144             delete e;
00145         }
00146
00147         /** @brief test link function
00148         */
00149         TEST_METHOD(link_right) {
00150             Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00151             Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00152
00153             Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(NULL, RightNode);
00154             LeftNode->Link(e);
00155             Assert::IsTrue(LeftNode == e->LeftNode());
00156             Assert::IsTrue(RightNode == e->RightNode());
00157             delete LeftNode;
00158             delete RightNode;
00159             delete e;
00160         }
00161
00162         /** @brief test RandomNext with low values
00163         */
00164         TEST_METHOD(rand_next_low) {
00165
00166             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00167             Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');

```

```

00168         Markov::Edge<unsigned char>* e = src->Link(target1);
00169         e->AdjustEdge(15);
00170         Markov::Node<unsigned char>* res = src->RandomNext();
00171         Assert::IsTrue(res == target1);
00172         delete src;
00173         delete target1;
00174         delete e;
00175     }
00176 }
00177
00178 /** @brief test RandomNext with 32 bit high values
00179 */
00180 TEST_METHOD(rand_next_u32) {
00181
00182     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00183     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00184     Markov::Edge<unsigned char>* e = src->Link(target1);
00185     e->AdjustEdge(1 << 31);
00186     Markov::Node<unsigned char>* res = src->RandomNext();
00187     Assert::IsTrue(res == target1);
00188     delete src;
00189     delete target1;
00190     delete e;
00191 }
00192
00193
00194 /** @brief random next on a node with no follow-ups
00195 */
00196 TEST_METHOD(rand_next_choice_1) {
00197
00198     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00199     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00200     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00201     Markov::Edge<unsigned char>* e1 = src->Link(target1);
00202     Markov::Edge<unsigned char>* e2 = src->Link(target2);
00203     e1->AdjustEdge(1);
00204     e2->AdjustEdge((unsigned long)(1ull << 31));
00205     Markov::Node<unsigned char>* res = src->RandomNext();
00206     Assert::IsNotNull(res);
00207     Assert::IsTrue(res == target2);
00208     delete src;
00209     delete target1;
00210     delete e1;
00211     delete e2;
00212 }
00213
00214 /** @brief random next on a node with no follow-ups
00215 */
00216 TEST_METHOD(rand_next_choice_2) {
00217
00218     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00219     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00220     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00221     Markov::Edge<unsigned char>* e1 = src->Link(target1);
00222     Markov::Edge<unsigned char>* e2 = src->Link(target2);
00223     e2->AdjustEdge(1);
00224     e1->AdjustEdge((unsigned long)(1ull << 31));
00225     Markov::Node<unsigned char>* res = src->RandomNext();
00226     Assert::IsNotNull(res);
00227     Assert::IsTrue(res == target1);
00228     delete src;
00229     delete target1;
00230     delete e1;
00231     delete e2;
00232 }
00233
00234
00235 /** @brief test updateEdges
00236 */
00237 TEST_METHOD(update_edges_count) {
00238
00239     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00240     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00241     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00242     Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00243     Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00244     e1->AdjustEdge(25);
00245     src->UpdateEdges(e1);
00246     e2->AdjustEdge(30);
00247     src->UpdateEdges(e2);
00248
00249     Assert::AreEqual((size_t)2, src->Edges()->size());
00250
00251     delete src;
00252     delete target1;
00253     delete e1;
00254     delete e2;

```

```

00255
00256     }
00257
00258     /** @brief test updateEdges
00259     */
00260     TEST_METHOD(update_edges_total) {
00261
00262         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00263         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00264         Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00265         Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target1);
00266         e1->AdjustEdge(25);
00267         src->UpdateEdges(e1);
00268         e2->AdjustEdge(30);
00269         src->UpdateEdges(e2);
00270
00271         Assert::AreEqual(55ull, src->TotalEdgeWeights());
00272
00273         delete src;
00274         delete target1;
00275         delete e1;
00276         delete e2;
00277     }
00278
00279
00280     /** @brief test FindVertice
00281     */
00282     TEST_METHOD(find_vertice) {
00283
00284         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00285         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00286         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00287         Markov::Edge<unsigned char>* res = NULL;
00288         src->Link(target1);
00289         src->Link(target2);
00290
00291
00292         res = src->FindEdge('b');
00293         Assert::IsNotNull(res);
00294         Assert::AreEqual((unsigned char)'b', res->TraverseNode()->NodeValue());
00295         res = src->FindEdge('c');
00296         Assert::IsNotNull(res);
00297         Assert::AreEqual((unsigned char)'c', res->TraverseNode()->NodeValue());
00298
00299         delete src;
00300         delete target1;
00301         delete target2;
00302
00303     }
00304
00305
00306
00307     /** @brief test FindVertice
00308     */
00309     TEST_METHOD(find_vertice_without_any) {
00310
00311         auto _invalid_next = [] {
00312             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00313             Markov::Edge<unsigned char>* res = NULL;
00314
00315             res = src->FindEdge('b');
00316             Assert::IsNull(res);
00317
00318             delete src;
00319         };
00320
00321         //Assert::ExpectException<std::logic_error>(_invalid_next);
00322     }
00323
00324     /** @brief test FindVertice
00325     */
00326     TEST_METHOD(find_vertice_nonexistent) {
00327
00328         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00329         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00330         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00331         Markov::Edge<unsigned char>* res = NULL;
00332         src->Link(target1);
00333         src->Link(target2);
00334
00335         res = src->FindEdge('D');
00336         Assert::IsNull(res);
00337
00338         delete src;
00339         delete target1;
00340         delete target2;
00341

```

```

00342     }
00343 };
00344
00345 /** @brief Test class for minimal viable Model
00346 */
00347 TEST_CLASS(Model)
00348 {
00349 public:
00350     /** @brief test model constructor for starter node
00351     */
00352     TEST_METHOD(model_constructor) {
00353         Markov::Model<unsigned char> m;
00354         Assert::AreEqual((unsigned char)'0', m.StarterNode()->NodeValue());
00355     }
00356
00357     /** @brief test import
00358     */
00359     TEST_METHOD(import_filename) {
00360         Markov::Model<unsigned char> m;
00361         Assert::IsTrue(m.Import("../MarkovPasswords/Models/2gram.mdl"));
00362     }
00363
00364     /** @brief test export
00365     */
00366     TEST_METHOD(export_filename) {
00367         Markov::Model<unsigned char> m;
00368         Assert::IsTrue(m.Export("../MarkovPasswords/Models/testcase.mdl"));
00369     }
00370
00371     /** @brief test random walk
00372     */
00373     TEST_METHOD(random_walk) {
00374         Markov::Model<unsigned char> m;
00375         Assert::IsTrue(m.Import("../models/finished.mdl"));
00376         Assert::IsNotNull(m.RandomWalk(1,12));
00377     }
00378 };
00379 }
00380
00381 /** @brief Testing namespace for MVP MarkovPasswords
00382 */
00383 namespace MarkovPasswords
00384 {
00385     /** @brief Test Class for Argparse class
00386     */
00387     TEST_CLASS(ArgParser)
00388     {
00389     public:
00390         /** @brief test basic generate
00391         */
00392         TEST_METHOD(generate_basic) {
00393             int argc = 8;
00394             char *argv[] = {"markov.exe", "generate", "-if", "model.mdl", "-of",
"passwords.txt", "-n", "100"};
00395
00396             /*ProgramOptions *p = Argparse::parse(argc, argv);
00397             Assert::IsNotNull(p);
00398
00399             Assert::AreEqual(p->bImport, true);
00400             Assert::AreEqual(p->bExport, false);
00401             Assert::AreEqual(p->importname, "model.mdl");
00402             Assert::AreEqual(p->outputfilename, "passwords.txt");
00403             Assert::AreEqual(p->generateN, 100); */
00404
00405         }
00406
00407         /** @brief test basic generate reordered params
00408         */
00409         TEST_METHOD(generate_basic_reorder) {
00410             int argc = 8;
00411             char *argv[] = { "markov.exe", "generate", "-n", "100", "-if", "model.mdl", "-of",
"passwords.txt" };
00412
00413             /*ProgramOptions* p = Argparse::parse(argc, argv);
00414             Assert::IsNotNull(p);
00415
00416             Assert::AreEqual(p->bImport, true);
00417             Assert::AreEqual(p->bExport, false);
00418             Assert::AreEqual(p->importname, "model.mdl");
00419             Assert::AreEqual(p->outputfilename, "passwords.txt");
00420             Assert::AreEqual(p->generateN, 100);*/
00421
00422         }
00423
00424         /** @brief test basic generate param longnames
00425         */
00426         TEST_METHOD(generate_basic_longname) {
00427             int argc = 8;

```

```

00427         char *argv[] = { "markov.exe", "generate", "-n", "100", "--inputfilename",
00428 "model.mdl", "--outputfilename", "passwords.txt" };
00429
00430         /*ProgramOptions* p = Argparse::parse(argc, argv);
00431         Assert::IsNotNull(p);
00432
00433         Assert::AreEqual(p->bImport, true);
00434         Assert::AreEqual(p->bExport, false);
00435         Assert::AreEqual(p->importname, "model.mdl");
00436         Assert::AreEqual(p->outputfilename, "passwords.txt");
00437         Assert::AreEqual(p->generateN, 100); */
00438     }
00439
00440     /** @brief test basic generate
00441     */
00442     TEST_METHOD(generate_fail_badmethod) {
00443         int argc = 8;
00444         char *argv[] = { "markov.exe", "junk", "-n", "100", "--inputfilename",
00445 "model.mdl", "--outputfilename", "passwords.txt" };
00446
00447         /*ProgramOptions* p = Argparse::parse(argc, argv);
00448         Assert::IsNotNull(p); */
00449     }
00450
00451     /** @brief test basic generate
00452     */
00453     TEST_METHOD(train_basic) {
00454         int argc = 4;
00455         char *argv[] = { "markov.exe", "train", "-ef", "model.mdl" };
00456
00457         /*ProgramOptions* p = Argparse::parse(argc, argv);
00458         Assert::IsNotNull(p);
00459
00460         Assert::AreEqual(p->bImport, false);
00461         Assert::AreEqual(p->bExport, true);
00462         Assert::AreEqual(p->exportname, "model.mdl"); */
00463     }
00464
00465     /** @brief test basic generate
00466     */
00467     TEST_METHOD(train_basic_longname) {
00468         int argc = 4;
00469         char *argv[] = { "markov.exe", "train", "--exportfilename", "model.mdl" };
00470
00471         /*ProgramOptions* p = Argparse::parse(argc, argv);
00472         Assert::IsNotNull(p);
00473
00474         Assert::AreEqual(p->bImport, false);
00475         Assert::AreEqual(p->bExport, true);
00476         Assert::AreEqual(p->exportname, "model.mdl"); */
00477     }
00478
00479     };
00480
00481     };
00482
00483     };
00484
00485
00486     /** @brief Testing namespace for MarkovModel
00487     */
00488     namespace MarkovModel {
00489
00490         /** @brief Test class for rest of Edge cases
00491         */
00492         TEST_CLASS(Edge)
00493         {
00494         public:
00495             /** @brief send exception on integer underflow
00496             */
00497             TEST_METHOD(except_integer_underflow) {
00498                 auto _underflow_adjust = [] {
00499                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00500                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00501                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00502 RightNode);
00503                     e->AdjustEdge(15);
00504                     e->AdjustEdge(-30);
00505                     delete LeftNode;
00506                     delete RightNode;
00507                     delete e;
00508                 };
00509                 Assert::ExpectException<std::underflow_error>(_underflow_adjust);
00510             }
00511         };
00512     };

```

```

00511         /** @brief test integer overflows
00512         */
00513         TEST_METHOD(except_integer_overflow) {
00514             auto _overflow_adjust = [] {
00515                 Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00516                 Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00517                 Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
RightNode);
00518                 e->AdjustEdge(~0ull);
00519                 e->AdjustEdge(1);
00520                 delete LeftNode;
00521                 delete RightNode;
00522                 delete e;
00523             };
00524             Assert::ExpectException<std::underflow_error>(_overflow_adjust);
00525         }
00526     };
00527
00528     /** @brief Test class for rest of Node cases
00529     */
00530     TEST_CLASS(Node)
00531     {
00532     public:
00533
00534         /** @brief test RandomNext with 64 bit high values
00535         */
00536         TEST_METHOD(rand_next_u64) {
00537
00538             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00539             Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00540             Markov::Edge<unsigned char>* e = src->Link(target1);
00541             e->AdjustEdge((unsigned long)(1ull << 63));
00542             Markov::Node<unsigned char>* res = src->RandomNext();
00543             Assert::IsTrue(res == target1);
00544             delete src;
00545             delete target1;
00546             delete e;
00547
00548         }
00549
00550         /** @brief test RandomNext with 64 bit high values
00551         */
00552         TEST_METHOD(rand_next_u64_max) {
00553
00554             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00555             Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00556             Markov::Edge<unsigned char>* e = src->Link(target1);
00557             e->AdjustEdge((0xffffffff));
00558             Markov::Node<unsigned char>* res = src->RandomNext();
00559             Assert::IsTrue(res == target1);
00560             delete src;
00561             delete target1;
00562             delete e;
00563
00564         }
00565
00566         /** @brief randomNext when no edges are present
00567         */
00568         TEST_METHOD(uninitialized_rand_next) {
00569
00570             auto _invalid_next = [] {
00571                 Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00572                 Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00573                 Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(src, target1);
00574                 Markov::Node<unsigned char>* res = src->RandomNext();
00575
00576                 delete src;
00577                 delete target1;
00578                 delete e;
00579             };
00580
00581             Assert::ExpectException<std::logic_error>(_invalid_next);
00582         }
00583
00584     };
00585
00586     /** @brief Test class for rest of model cases
00587     */
00588     TEST_CLASS(Model)
00589     {
00590     public:
00591
00592         TEST_METHOD(functional_random_walk) {
00593             Markov::Model<unsigned char> m;
00594             Markov::Node<unsigned char>* starter = m.StarterNode();
00595             Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00596             Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');

```

```

00597         Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00598         Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
00599         starter->Link(a)->AdjustEdge(1);
00600         a->Link(b)->AdjustEdge(1);
00601         b->Link(c)->AdjustEdge(1);
00602         c->Link(end)->AdjustEdge(1);
00603
00604         char* res = (char*)m.RandomWalk(1,12);
00605         Assert::IsFalse(strcmp(res, "abc"));
00606     }
00607     TEST_METHOD(functional_random_walk_without_any) {
00608         Markov::Model<unsigned char> m;
00609         Markov::Node<unsigned char>* starter = m.StarterNode();
00610         Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00611         Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
00612         Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00613         Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
00614         Markov::Edge<unsigned char>* res = NULL;
00615         starter->Link(a)->AdjustEdge(1);
00616         a->Link(b)->AdjustEdge(1);
00617         b->Link(c)->AdjustEdge(1);
00618         c->Link(end)->AdjustEdge(1);
00619
00620         res = starter->FindEdge('D');
00621         Assert::IsNull(res);
00622     }
00623 };
00624 };
00625
00626 }
00627
00628 /** @brief Testing namespace for MarkovPasswords
00629 */
00630 namespace MarkovPasswords {
00631
00632 };
00633
00634 }

```


Index

- [_left](#)
 - [Markov::Edge< NodeStorageType >, 84](#)
 - [_right](#)
 - [Markov::Edge< NodeStorageType >, 84](#)
 - [_value](#)
 - [Markov::Node< storageType >, 153](#)
 - [_weight](#)
 - [Markov::Edge< NodeStorageType >, 84](#)
- [about](#)
 - [Markov::GUI::about, 43](#)
 - [Markov::GUI::CLI, 50](#)
 - [Markov::GUI::menu, 110](#)
- [about.h, 165](#)
- [action](#)
 - [markopy_cli, 19](#)
- [AdjustEdge](#)
 - [Markov::API::CUDA::CUDAModelMatrix, 57](#)
 - [Markov::API::MarkovPasswords, 88](#)
 - [Markov::API::ModelMatrix, 128](#)
 - [Markov::Edge< NodeStorageType >, 81](#)
 - [Markov::Model< NodeStorageType >, 117](#)
- [alphabet](#)
 - [model_2gram, 24](#)
 - [random-model, 25](#)
- [Argparse](#)
 - [Markov::API::CLI::Argparse, 45](#)
- [argparse.cpp, 166](#)
- [argparse.h, 166, 168](#)
 - [BOOST_ALL_DYN_LINK, 168](#)
- [args](#)
 - [markopy_cli, 19](#)
- [bExport](#)
 - [Markov::API::CLI::_programOptions, 41](#)
- [bFailure](#)
 - [Markov::API::CLI::_programOptions, 41](#)
- [blImport](#)
 - [Markov::API::CLI::_programOptions, 41](#)
- [BLACK](#)
 - [Markov::API::CLI::Terminal, 157](#)
- [BLUE](#)
 - [Markov::API::CLI::Terminal, 157](#)
- [BOOST_ALL_DYN_LINK](#)
 - [argparse.h, 168](#)
- [BOOST_PYTHON_MODULE](#)
 - [Markov::Markopy, 23](#)
- [BOOST_PYTHON_STATIC_LIB](#)
 - [markopy.cpp, 183](#)
- [BROWN](#)
 - [Markov::API::CLI::Terminal, 157](#)
- [bulk](#)
 - [markopy_cli, 19](#)
- [CLI](#)
 - [Markov::GUI::CLI, 50](#)
- [CLI.h, 170, 171](#)
- [cli_generate](#)
 - [markopy_cli, 17](#)
- [cli_init](#)
 - [markopy_cli, 18](#)
- [cli_train](#)
 - [markopy_cli, 18](#)
- [color](#)
 - [Markov::API::CLI::Terminal, 157](#)
- [colormap](#)
 - [Markov::API::CLI::Terminal, 157](#)
- [ConstructMatrix](#)
 - [Markov::API::CUDA::CUDAModelMatrix, 58](#)
 - [Markov::API::ModelMatrix, 129](#)
- [corpus_list](#)
 - [markopy_cli, 19](#)
- [CudaCheckNotifyErr](#)
 - [Markov::API::CUDA::CUDADeviceController, 53](#)
 - [Markov::API::CUDA::CUDAModelMatrix, 59](#)
- [cudaDeviceController.h, 172](#)
- [cudaModelMatrix.h, 173, 174](#)
- [CYAN](#)
 - [Markov::API::CLI::Terminal, 157](#)
- [DARKGRAY](#)
 - [Markov::API::CLI::Terminal, 157](#)
- [datasetFile](#)
 - [Markov::API::CUDA::CUDAModelMatrix, 73](#)
 - [Markov::API::MarkovPasswords, 98](#)
 - [Markov::API::ModelMatrix, 144](#)
- [datasetname](#)
 - [Markov::API::CLI::_programOptions, 41](#)
- [default](#)
 - [markopy_cli, 19](#)
- [device_edgeMatrix](#)
 - [Markov::API::CUDA::CUDAModelMatrix, 73](#)
- [device_matrixIndex](#)
 - [Markov::API::CUDA::CUDAModelMatrix, 73](#)
- [device_totalEdgeWeights](#)
 - [Markov::API::CUDA::CUDAModelMatrix, 73](#)
- [device_valueMatrix](#)
 - [Markov::API::CUDA::CUDAModelMatrix, 73](#)
- [distribution](#)
 - [Markov::Random::DefaultRandomEngine, 77](#)

- Markov::Random::Marsaglia, 105
 - Markov::Random::Mersenne, 113
- dllmain.cpp, 174, 175
- DumpJSON
 - Markov::API::CUDA::CUDAModelMatrix, 60
 - Markov::API::ModelMatrix, 130
- Edge
 - Markov::Edge< NodeStorageType >, 81
- edge.h, 175, 176
- edgeMatrix
 - Markov::API::CUDA::CUDAModelMatrix, 73
 - Markov::API::ModelMatrix, 144
- Edges
 - Markov::API::CUDA::CUDAModelMatrix, 60
 - Markov::API::MarkovPasswords, 89
 - Markov::API::ModelMatrix, 131
 - Markov::Model< NodeStorageType >, 118
 - Markov::Node< storageType >, 149
- edges
 - Markov::API::CUDA::CUDAModelMatrix, 74
 - Markov::API::MarkovPasswords, 98
 - Markov::API::ModelMatrix, 144
 - Markov::Model< NodeStorageType >, 124
 - Markov::Node< storageType >, 153
- edgesV
 - Markov::Node< storageType >, 153
- EdgeWeight
 - Markov::Edge< NodeStorageType >, 82
- endl
 - Markov::API::CLI::Terminal, 158
- Export
 - Markov::API::CUDA::CUDAModelMatrix, 60, 61
 - Markov::API::MarkovPasswords, 89
 - Markov::API::ModelMatrix, 131
 - Markov::Model< NodeStorageType >, 118, 119
- exportname
 - Markov::API::CLI::_programOptions, 41
- f
 - model_2gram, 25
 - random-model, 25
- FastRandomWalk
 - Markov::API::CUDA::CUDAModelMatrix, 61
 - Markov::API::ModelMatrix, 132
- FastRandomWalkPartition
 - Markov::API::CUDA::CUDAModelMatrix, 62
 - Markov::API::ModelMatrix, 133
- FastRandomWalkThread
 - Markov::API::CUDA::CUDAModelMatrix, 63
 - Markov::API::ModelMatrix, 134
- FindEdge
 - Markov::Node< storageType >, 149
- framework.h, 178, 179
 - WIN32_LEAN_AND_MEAN, 179
- Generate
 - Markov::API::CUDA::CUDAModelMatrix, 64
 - Markov::API::MarkovPasswords, 90
 - Markov::API::ModelMatrix, 136
- generateN
 - Markov::API::CLI::_programOptions, 41
- GenerateThread
 - Markov::API::CUDA::CUDAModelMatrix, 65
 - Markov::API::MarkovPasswords, 91
 - Markov::API::ModelMatrix, 137
- generator
 - Markov::Random::DefaultRandomEngine, 77
 - Markov::Random::Marsaglia, 106
 - Markov::Random::Mersenne, 113
- getProgramOptions
 - Markov::API::CLI::Argparse, 47
- GREEN
 - Markov::API::CLI::Terminal, 157
- help
 - markopy_cli, 19
 - Markov::API::CLI::Argparse, 47
- home
 - Markov::GUI::Train, 163
- Import
 - Markov::API::CUDA::CUDAModelMatrix, 66, 67
 - Markov::API::MarkovPasswords, 92
 - Markov::API::ModelMatrix, 138
 - Markov::Model< NodeStorageType >, 120
- importname
 - Markov::API::CLI::_programOptions, 41
- LeftNode
 - Markov::Edge< NodeStorageType >, 82
- LIGHTGRAY
 - Markov::API::CLI::Terminal, 157
- Link
 - Markov::Node< storageType >, 150
- ListCudaDevices
 - Markov::API::CUDA::CUDADeviceController, 53
 - Markov::API::CUDA::CUDAModelMatrix, 68
- listfile
 - Markov::API::Concurrency::ThreadSharedListHandler, 161
- loadDataset
 - Markov::GUI::MarkovPasswordsGUI, 101
- MAGENTA
 - Markov::API::CLI::Terminal, 157
- main
 - src/main.cpp, 179
 - UI/src/main.cpp, 181
- main.cpp, 179–182
- markopy.cpp, 182, 183
 - BOOST_PYTHON_STATIC_LIB, 183
- markopy_cli, 17
 - action, 19
 - args, 19
 - bulk, 19
 - cli_generate, 17
 - cli_init, 18

- cli_train, 18
- corpus_list, 19
- default, 19
- help, 19
- model, 19
- model_base, 19
- model_extension, 19
- model_list, 20
- output, 20
- output_file_name, 20
- output_forced, 20
- parser, 20
- True, 20
- markopy_cli.py, 184
- Markov, 20
- Markov::API, 21
- Markov::API::CLI, 21
 - operator<<, 22
 - ProgramOptions, 22
- Markov::API::CLI::_programOptions, 39
 - bExport, 41
 - bFailure, 41
 - blImport, 41
 - datasetname, 41
 - exportname, 41
 - generateN, 41
 - importname, 41
 - outputfilename, 41
 - seperator, 42
 - wordlistname, 42
- Markov::API::CLI::Argparse, 44
 - Argparse, 45
 - getProgramOptions, 47
 - help, 47
 - parse, 48
 - po, 49
 - setProgramOptions, 48
- Markov::API::CLI::Terminal, 155
 - BLACK, 157
 - BLUE, 157
 - BROWN, 157
 - color, 157
 - colormap, 157
 - CYAN, 157
 - DARKGRAY, 157
 - endl, 158
 - GREEN, 157
 - LIGHTGRAY, 157
 - MAGENTA, 157
 - RED, 157
 - RESET, 157
 - Terminal, 157
 - WHITE, 157
 - YELLOW, 157
- Markov::API::Concurrency, 22
- Markov::API::Concurrency::ThreadSharedListHandler, 158
 - listfile, 161
 - mlock, 161
 - next, 161
 - ThreadSharedListHandler, 160
- Markov::API::CUDA, 22
- Markov::API::CUDA::CUDADeviceController, 51
 - CudaCheckNotifyErr, 53
 - ListCudaDevices, 53
- Markov::API::CUDA::CUDAModelMatrix, 53
 - AdjustEdge, 57
 - ConstructMatrix, 58
 - CudaCheckNotifyErr, 59
 - datasetFile, 73
 - device_edgeMatrix, 73
 - device_matrixIndex, 73
 - device_totalEdgeWeights, 73
 - device_valueMatrix, 73
 - DumpJSON, 60
 - edgeMatrix, 73
 - Edges, 60
 - edges, 74
 - Export, 60, 61
 - FastRandomWalk, 61
 - FastRandomWalkPartition, 62
 - FastRandomWalkThread, 63
 - Generate, 64
 - GenerateThread, 65
 - Import, 66, 67
 - ListCudaDevices, 68
 - matrixIndex, 74
 - matrixSize, 74
 - MigrateMatrix, 68
 - modelSavefile, 74
 - Nodes, 68
 - nodes, 74
 - OpenDatasetFile, 68
 - outputFile, 74
 - RandomWalk, 69
 - RetrieveCudaBuffer, 70
 - Save, 70
 - StarterNode, 71
 - starterNode, 74
 - totalEdgeWeights, 74
 - Train, 71
 - TrainThread, 72
 - valueMatrix, 74
- Markov::API::MarkovPasswords, 84
 - AdjustEdge, 88
 - datasetFile, 98
 - Edges, 89
 - edges, 98
 - Export, 89
 - Generate, 90
 - GenerateThread, 91
 - Import, 92
 - MarkovPasswords, 88
 - modelSavefile, 98
 - Nodes, 93
 - nodes, 98

- OpenDatasetFile, 93
- outputFile, 98
- RandomWalk, 94
- Save, 95
- StarterNode, 96
- starterNode, 98
- Train, 96
- TrainThread, 97
- Markov::API::ModelMatrix, 124
 - AdjustEdge, 128
 - ConstructMatrix, 129
 - datasetFile, 144
 - DumpJSON, 130
 - edgeMatrix, 144
 - Edges, 131
 - edges, 144
 - Export, 131
 - FastRandomWalk, 132
 - FastRandomWalkPartition, 133
 - FastRandomWalkThread, 134
 - Generate, 136
 - GenerateThread, 137
 - Import, 138
 - matrixIndex, 144
 - matrixSize, 144
 - ModelMatrix, 128
 - modelSavefile, 144
 - Nodes, 139
 - nodes, 145
 - OpenDatasetFile, 139
 - outputFile, 145
 - RandomWalk, 140
 - Save, 141
 - StarterNode, 142
 - starterNode, 145
 - totalEdgeWeights, 145
 - Train, 142
 - TrainThread, 143
 - valueMatrix, 145
- Markov::Edge< NodeStorageType >, 79
 - _left, 84
 - _right, 84
 - _weight, 84
 - AdjustEdge, 81
 - Edge, 81
 - EdgeWeight, 82
 - LeftNode, 82
 - RightNode, 82
 - SetLeftEdge, 83
 - SetRightEdge, 83
 - TraverseNode, 83
- Markov::GUI, 23
- Markov::GUI::about, 42
 - about, 43
 - ui, 43
- Markov::GUI::CLI, 49
 - about, 50
 - CLI, 50
 - start, 51
 - statistics, 51
 - ui, 51
- Markov::GUI::MarkovPasswordsGUI, 99
 - loadDataset, 101
 - MarkovPasswordsGUI, 101
 - MarkovPasswordsGUI::benchmarkSelected, 101
 - MarkovPasswordsGUI::comparisonSelected, 101
 - MarkovPasswordsGUI::modelvisSelected, 101
 - MarkovPasswordsGUI::visualDebugSelected, 102
 - renderHTMLFile, 102
 - ui, 102
- Markov::GUI::menu, 108
 - about, 110
 - menu, 109
 - ui, 110
 - visualization, 110
- Markov::GUI::Train, 162
 - home, 163
 - Train, 163
 - train, 163
 - ui, 163
- Markov::Markopy, 23
 - BOOST_PYTHON_MODULE, 23
- Markov::Model< NodeStorageType >, 115
 - AdjustEdge, 117
 - Edges, 118
 - edges, 124
 - Export, 118, 119
 - Import, 120
 - Model, 117
 - Nodes, 122
 - nodes, 124
 - RandomWalk, 122
 - StarterNode, 123
 - starterNode, 124
- Markov::Node< storageType >, 145
 - _value, 153
 - Edges, 149
 - edges, 153
 - edgesV, 153
 - FindEdge, 149
 - Link, 150
 - Node, 148
 - NodeValue, 151
 - RandomNext, 151
 - total_edge_weights, 153
 - TotalEdgeWeights, 152
 - UpdateEdges, 152
 - UpdateTotalVerticeWeight, 153
- Markov::Random, 24
- Markov::Random::DefaultRandomEngine, 75
 - distribution, 77
 - generator, 77
 - random, 78
 - rd, 78
- Markov::Random::Marsaglia, 102
 - distribution, 105

- generator, 106
- Marsaglia, 105
- random, 106
- rd, 107
- x, 107
- y, 107
- z, 108
- Markov::Random::Mersenne, 110
 - distribution, 113
 - generator, 113
 - random, 114
 - rd, 114
- Markov::Random::RandomEngine, 154
 - random, 155
- MarkovPasswords
 - Markov::API::MarkovPasswords, 88
- markovPasswords.cpp, 187
- markovPasswords.h, 189, 190
- MarkovPasswordsGUI
 - Markov::GUI::MarkovPasswordsGUI, 101
- MarkovPasswordsGUI.cpp, 191
- MarkovPasswordsGUI.h, 192, 193
- MarkovPasswordsGUI::benchmarkSelected
 - Markov::GUI::MarkovPasswordsGUI, 101
- MarkovPasswordsGUI::comparisonSelected
 - Markov::GUI::MarkovPasswordsGUI, 101
- MarkovPasswordsGUI::modelvisSelected
 - Markov::GUI::MarkovPasswordsGUI, 101
- MarkovPasswordsGUI::visualDebugSelected
 - Markov::GUI::MarkovPasswordsGUI, 102
- Marsaglia
 - Markov::Random::Marsaglia, 105
- matrixIndex
 - Markov::API::CUDA::CUDAModelMatrix, 74
 - Markov::API::ModelMatrix, 144
- matrixSize
 - Markov::API::CUDA::CUDAModelMatrix, 74
 - Markov::API::ModelMatrix, 144
- menu
 - Markov::GUI::menu, 109
- menu.cpp, 194
- menu.h, 195
- MigrateMatrix
 - Markov::API::CUDA::CUDAModelMatrix, 68
- mlock
 - Markov::API::Concurrency::ThreadSharedListHandler, 161
- Model
 - Markov::Model< NodeStorageType >, 117
- model
 - markopy_cli, 19
- model.h, 196, 197
- model_2gram, 24
 - alphabet, 24
 - f, 25
- model_2gram.py, 201
- model_base
 - markopy_cli, 19
- model_extension
 - markopy_cli, 19
- model_list
 - markopy_cli, 20
- ModelMatrix
 - Markov::API::ModelMatrix, 128
- modelMatrix.cpp, 202
- modelMatrix.h, 205
- modelSavefile
 - Markov::API::CUDA::CUDAModelMatrix, 74
 - Markov::API::MarkovPasswords, 98
 - Markov::API::ModelMatrix, 144
- next
 - Markov::API::Concurrency::ThreadSharedListHandler, 161
- Node
 - Markov::Node< storageType >, 148
- node.h, 207, 208
- Nodes
 - Markov::API::CUDA::CUDAModelMatrix, 68
 - Markov::API::MarkovPasswords, 93
 - Markov::API::ModelMatrix, 139
 - Markov::Model< NodeStorageType >, 122
- nodes
 - Markov::API::CUDA::CUDAModelMatrix, 74
 - Markov::API::MarkovPasswords, 98
 - Markov::API::ModelMatrix, 145
 - Markov::Model< NodeStorageType >, 124
- NodeValue
 - Markov::Node< storageType >, 151
- OpenDatasetFile
 - Markov::API::CUDA::CUDAModelMatrix, 68
 - Markov::API::MarkovPasswords, 93
 - Markov::API::ModelMatrix, 139
- operator<<
 - Markov::API::CLI, 22
 - term.cpp, 219
- output
 - markopy_cli, 20
- output_file_name
 - markopy_cli, 20
- output_forced
 - markopy_cli, 20
- outputFile
 - Markov::API::CUDA::CUDAModelMatrix, 74
 - Markov::API::MarkovPasswords, 98
 - Markov::API::ModelMatrix, 145
- outputfilename
 - Markov::API::CLI::_programOptions, 41
- parse
 - Markov::API::CLI::Argparse, 48
- parser
 - markopy_cli, 20
- pch.cpp, 211, 212
- pch.h, 213, 214
- po

- Markov::API::CLI::Argparse, 49
- ProgramOptions
 - Markov::API::CLI, 22
- random, 25
 - Markov::Random::DefaultRandomEngine, 78
 - Markov::Random::Marsaglia, 106
 - Markov::Random::Mersenne, 114
 - Markov::Random::RandomEngine, 155
- random-model, 25
 - alphabet, 25
 - f, 25
- random-model.py, 214
- random.h, 215, 216
- RandomNext
 - Markov::Node< storageType >, 151
- RandomWalk
 - Markov::API::CUDA::CUDAModelMatrix, 69
 - Markov::API::MarkovPasswords, 94
 - Markov::API::ModelMatrix, 140
 - Markov::Model< NodeStorageType >, 122
- rd
 - Markov::Random::DefaultRandomEngine, 78
 - Markov::Random::Marsaglia, 107
 - Markov::Random::Mersenne, 114
- README.md, 218
- RED
 - Markov::API::CLI::Terminal, 157
- renderHTMLFile
 - Markov::GUI::MarkovPasswordsGUI, 102
- RESET
 - Markov::API::CLI::Terminal, 157
- RetrieveCudaBuffer
 - Markov::API::CUDA::CUDAModelMatrix, 70
- RightNode
 - Markov::Edge< NodeStorageType >, 82
- Save
 - Markov::API::CUDA::CUDAModelMatrix, 70
 - Markov::API::MarkovPasswords, 95
 - Markov::API::ModelMatrix, 141
- separator
 - Markov::API::CLI::_programOptions, 42
- SetLeftEdge
 - Markov::Edge< NodeStorageType >, 83
- setProgramOptions
 - Markov::API::CLI::Argparse, 48
- SetRightEdge
 - Markov::Edge< NodeStorageType >, 83
- src/main.cpp
 - main, 179
- start
 - Markov::GUI::CLI, 51
- StarterNode
 - Markov::API::CUDA::CUDAModelMatrix, 71
 - Markov::API::MarkovPasswords, 96
 - Markov::API::ModelMatrix, 142
 - Markov::Model< NodeStorageType >, 123
- starterNode
 - Markov::API::CUDA::CUDAModelMatrix, 74
 - Markov::API::MarkovPasswords, 98
 - Markov::API::ModelMatrix, 145
 - Markov::Model< NodeStorageType >, 124
- statistics
 - Markov::GUI::CLI, 51
- term.cpp, 218, 219
 - operator<<, 219
- term.h, 220, 222
 - TERM_FAIL, 221
 - TERM_INFO, 221
 - TERM_SUCC, 222
 - TERM_WARN, 222
- TERM_FAIL
 - term.h, 221
- TERM_INFO
 - term.h, 221
- TERM_SUCC
 - term.h, 222
- TERM_WARN
 - term.h, 222
- Terminal
 - Markov::API::CLI::Terminal, 157
- TEST_CLASS
 - Testing::MarkovModel, 26–28
 - Testing::MVP::MarkovModel, 30–32
 - Testing::MVP::MarkovPasswords, 35
- Testing, 25
 - Testing::MarkovModel, 26
 - TEST_CLASS, 26–28
 - Testing::MarkovPasswords, 29
 - Testing::MVP, 29
 - Testing::MVP::MarkovModel, 29
 - TEST_CLASS, 30–32
 - Testing::MVP::MarkovPasswords, 35
 - TEST_CLASS, 35
- ThreadSharedListHandler
 - Markov::API::Concurrency::ThreadSharedListHandler, 160
- threadSharedListHandler.cpp, 223
- threadSharedListHandler.h, 223, 224
- total_edge_weights
 - Markov::Node< storageType >, 153
- TotalEdgeWeights
 - Markov::Node< storageType >, 152
- totalEdgeWeights
 - Markov::API::CUDA::CUDAModelMatrix, 74
 - Markov::API::ModelMatrix, 145
- Train
 - Markov::API::CUDA::CUDAModelMatrix, 71
 - Markov::API::MarkovPasswords, 96
 - Markov::API::ModelMatrix, 142
 - Markov::GUI::Train, 163
- train
 - Markov::GUI::Train, 163
- Train.h, 226
- TrainThread
 - Markov::API::CUDA::CUDAModelMatrix, 72

- Markov::API::MarkovPasswords, [97](#)
- Markov::API::ModelMatrix, [143](#)
- TraverseNode
 - Markov::Edge< NodeStorageType >, [83](#)
- True
 - markopy_cli, [20](#)
- ui
 - Markov::GUI::about, [43](#)
 - Markov::GUI::CLI, [51](#)
 - Markov::GUI::MarkovPasswordsGUI, [102](#)
 - Markov::GUI::menu, [110](#)
 - Markov::GUI::Train, [163](#)
- UI/src/main.cpp
 - main, [181](#)
- UnitTests.cpp, [227](#), [228](#)
- UpdateEdges
 - Markov::Node< storageType >, [152](#)
- UpdateTotalVerticeWeight
 - Markov::Node< storageType >, [153](#)
- valueMatrix
 - Markov::API::CUDA::CUDAModelMatrix, [74](#)
 - Markov::API::ModelMatrix, [145](#)
- visualization
 - Markov::GUI::menu, [110](#)
- WHITE
 - Markov::API::CLI::Terminal, [157](#)
- WIN32_LEAN_AND_MEAN
 - framework.h, [179](#)
- wordlistname
 - Markov::API::CLI::_programOptions, [42](#)
- x
 - Markov::Random::Marsaglia, [107](#)
- y
 - Markov::Random::Marsaglia, [107](#)
- YELLOW
 - Markov::API::CLI::Terminal, [157](#)
- z
 - Markov::Random::Marsaglia, [108](#)