



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 Namespace Index	7
2.1 Namespace List	7
3 Hierarchical Index	9
3.1 Class Hierarchy	9
4 Class Index	11
4.1 Class List	11
5 File Index	13
5.1 File List	13
6 Namespace Documentation	15
6.1 markopy_cli Namespace Reference	15
6.1.1 Detailed Description	15
6.1.2 Function Documentation	15
6.1.2.1 cli_generate()	15
6.1.2.2 cli_init()	16
6.1.2.3 cli_train()	16
6.1.3 Variable Documentation	17
6.1.3.1 action	17

6.1.3.2 args	17
6.1.3.3 bulk	17
6.1.3.4 corpus_list	17
6.1.3.5 default	17
6.1.3.6 help	17
6.1.3.7 model	17
6.1.3.8 model_base	17
6.1.3.9 model_extension	18
6.1.3.10 model_list	18
6.1.3.11 output	18
6.1.3.12 output_file_name	18
6.1.3.13 output_forced	18
6.1.3.14 parser	18
6.1.3.15 True	18
6.2 Markov Namespace Reference	18
6.2.1 Detailed Description	19
6.3 Markov::API Namespace Reference	19
6.3.1 Detailed Description	19
6.4 Markov::API::CLI Namespace Reference	19
6.4.1 Detailed Description	20
6.4.2 Typedef Documentation	20
6.4.2.1 ProgramOptions	20
6.4.3 Function Documentation	20
6.4.3.1 operator<<()	20
6.5 Markov::API::Concurrency Namespace Reference	20
6.5.1 Detailed Description	20
6.6 Markov::GUI Namespace Reference	20
6.6.1 Detailed Description	21
6.7 Markov::Markopy Namespace Reference	21
6.7.1 Function Documentation	21
6.7.1.1 BOOST_PYTHON_MODULE()	21
6.8 Markov::Random Namespace Reference	21
6.8.1 Detailed Description	22
6.9 model_2gram Namespace Reference	22
6.9.1 Detailed Description	22
6.9.2 Variable Documentation	22
6.9.2.1 alphabet	22
6.9.2.2 f	22
6.10 random Namespace Reference	22
6.10.1 Detailed Description	22
6.11 random-model Namespace Reference	23
6.11.1 Variable Documentation	23

6.11.1.1 alphabet	23
6.11.1.2 f	23
6.12 Testing Namespace Reference	23
6.12.1 Detailed Description	23
6.13 Testing::MarkovModel Namespace Reference	23
6.13.1 Detailed Description	24
6.13.2 Function Documentation	24
6.13.2.1 TEST_CLASS() [1/3]	24
6.13.2.2 TEST_CLASS() [2/3]	24
6.13.2.3 TEST_CLASS() [3/3]	25
6.14 Testing::MarkovPasswords Namespace Reference	26
6.14.1 Detailed Description	26
6.15 Testing::MVP Namespace Reference	26
6.15.1 Detailed Description	27
6.16 Testing::MVP::MarkovModel Namespace Reference	27
6.16.1 Detailed Description	27
6.16.2 Function Documentation	27
6.16.2.1 TEST_CLASS() [1/3]	27
6.16.2.2 TEST_CLASS() [2/3]	29
6.16.2.3 TEST_CLASS() [3/3]	30
6.17 Testing::MVP::MarkovPasswords Namespace Reference	33
6.17.1 Detailed Description	33
6.17.2 Function Documentation	33
6.17.2.1 TEST_CLASS()	34
7 Class Documentation	37
7.1 Markov::API::CLI::_programOptions Struct Reference	37
7.1.1 Detailed Description	39
7.1.2 Member Data Documentation	39
7.1.2.1 bExport	39
7.1.2.2 bFailure	39
7.1.2.3 blmport	39
7.1.2.4 datasetname	39
7.1.2.5 exportname	39
7.1.2.6 generateN	39
7.1.2.7 importname	39
7.1.2.8 outputfilename	40
7.1.2.9 seperator	40
7.1.2.10 wordlistname	40
7.2 Markov::GUI::about Class Reference	40
7.2.1 Detailed Description	41
7.2.2 Constructor & Destructor Documentation	41

7.2.2.1 about()	41
7.2.3 Member Data Documentation	41
7.2.3.1 ui	41
7.3 Markov::API::CLI::Argparse Class Reference	42
7.3.1 Detailed Description	43
7.3.2 Constructor & Destructor Documentation	43
7.3.2.1 Argparse() [1/2]	43
7.3.2.2 Argparse() [2/2]	43
7.3.3 Member Function Documentation	45
7.3.3.1 getProgramOptions()	45
7.3.3.2 help()	45
7.3.3.3 parse()	45
7.3.3.4 setProgramOptions()	46
7.3.4 Member Data Documentation	47
7.3.4.1 po	47
7.4 Markov::GUI::CLI Class Reference	47
7.4.1 Detailed Description	48
7.4.2 Constructor & Destructor Documentation	48
7.4.2.1 CLI()	48
7.4.3 Member Function Documentation	48
7.4.3.1 about	49
7.4.3.2 start	49
7.4.3.3 statistics	49
7.4.4 Member Data Documentation	49
7.4.4.1 ui	49
7.5 Markov::Random::DefaultRandomEngine Class Reference	49
7.5.1 Detailed Description	51
7.5.2 Member Function Documentation	51
7.5.2.1 distribution()	51
7.5.2.2 generator()	52
7.5.2.3 random()	52
7.5.2.4 rd()	53
7.6 Markov::Edge< NodeStorageType > Class Template Reference	53
7.6.1 Detailed Description	56
7.6.2 Constructor & Destructor Documentation	56
7.6.2.1 Edge() [1/2]	56
7.6.2.2 Edge() [2/2]	56
7.6.3 Member Function Documentation	56
7.6.3.1 AdjustEdge()	57
7.6.3.2 EdgeWeight()	57
7.6.3.3 LeftNode()	58
7.6.3.4 RightNode()	58

7.6.3.5 SetLeftEdge()	59
7.6.3.6 SetRightEdge()	59
7.6.3.7 TraverseNode()	60
7.6.4 Member Data Documentation	61
7.6.4.1 _left	61
7.6.4.2 _right	61
7.6.4.3 _weight	61
7.7 Markov::API::MarkovPasswords Class Reference	61
7.7.1 Detailed Description	64
7.7.2 Constructor & Destructor Documentation	65
7.7.2.1 MarkovPasswords() [1/2]	65
7.7.2.2 MarkovPasswords() [2/2]	65
7.7.3 Member Function Documentation	65
7.7.3.1 AdjustEdge()	65
7.7.3.2 Edges()	66
7.7.3.3 Export() [1/2]	66
7.7.3.4 Export() [2/2]	66
7.7.3.5 Generate()	67
7.7.3.6 GenerateThread()	68
7.7.3.7 Import() [1/2]	69
7.7.3.8 Import() [2/2]	69
7.7.3.9 Nodes()	70
7.7.3.10 OpenDatasetFile()	70
7.7.3.11 RandomWalk()	71
7.7.3.12 Save()	72
7.7.3.13 StarterNode()	72
7.7.3.14 Train()	72
7.7.3.15 TrainThread()	73
7.7.4 Member Data Documentation	74
7.7.4.1 datasetFile	74
7.7.4.2 edges	74
7.7.4.3 modelSavefile	74
7.7.4.4 nodes	74
7.7.4.5 outputFile	75
7.7.4.6 starterNode	75
7.8 Markov::GUI::MarkovPasswordsGUI Class Reference	75
7.8.1 Detailed Description	77
7.8.2 Constructor & Destructor Documentation	77
7.8.2.1 MarkovPasswordsGUI()	77
7.8.3 Member Function Documentation	77
7.8.3.1 loadDataset()	77
7.8.3.2 MarkovPasswordsGUI::benchmarkSelected	77

7.8.3.3 MarkovPasswordsGUI::comparisonSelected	77
7.8.3.4 MarkovPasswordsGUI::modelvisSelected	78
7.8.3.5 MarkovPasswordsGUI::visualDebugSelected	78
7.8.3.6 renderHTMLFile()	78
7.8.4 Member Data Documentation	78
7.8.4.1 ui	78
7.9 Markov::Random::Marsaglia Class Reference	78
7.9.1 Detailed Description	81
7.9.2 Constructor & Destructor Documentation	81
7.9.2.1 Marsaglia()	81
7.9.3 Member Function Documentation	81
7.9.3.1 distribution()	81
7.9.3.2 generator()	82
7.9.3.3 random()	82
7.9.3.4 rd()	83
7.9.4 Member Data Documentation	83
7.9.4.1 x	83
7.9.4.2 y	83
7.9.4.3 z	83
7.10 Markov::GUI::menu Class Reference	84
7.10.1 Detailed Description	85
7.10.2 Constructor & Destructor Documentation	85
7.10.2.1 menu()	85
7.10.3 Member Function Documentation	85
7.10.3.1 about	85
7.10.3.2 visualization	85
7.10.4 Member Data Documentation	85
7.10.4.1 ui	86
7.11 Markov::Random::Mersenne Class Reference	86
7.11.1 Detailed Description	87
7.11.2 Member Function Documentation	88
7.11.2.1 distribution()	88
7.11.2.2 generator()	88
7.11.2.3 random()	89
7.11.2.4 rd()	89
7.12 Markov::Model< NodeStorageType > Class Template Reference	90
7.12.1 Detailed Description	92
7.12.2 Constructor & Destructor Documentation	92
7.12.2.1 Model()	92
7.12.3 Member Function Documentation	92
7.12.3.1 AdjustEdge()	93
7.12.3.2 Edges()	93

7.12.3.3 Export() [1/2]	93
7.12.3.4 Export() [2/2]	94
7.12.3.5 Import() [1/2]	94
7.12.3.6 Import() [2/2]	95
7.12.3.7 Nodes()	96
7.12.3.8 RandomWalk()	96
7.12.3.9 StarterNode()	98
7.12.4 Member Data Documentation	98
7.12.4.1 edges	98
7.12.4.2 nodes	98
7.12.4.3 starterNode	98
7.13 Markov::Node< storageType > Class Template Reference	99
7.13.1 Detailed Description	101
7.13.2 Constructor & Destructor Documentation	101
7.13.2.1 Node() [1/2]	101
7.13.2.2 Node() [2/2]	101
7.13.3 Member Function Documentation	102
7.13.3.1 Edges()	102
7.13.3.2 FindEdge() [1/2]	102
7.13.3.3 FindEdge() [2/2]	102
7.13.3.4 Link() [1/2]	103
7.13.3.5 Link() [2/2]	104
7.13.3.6 NodeValue()	105
7.13.3.7 RandomNext()	105
7.13.3.8 TotalEdgeWeights()	106
7.13.3.9 UpdateEdges()	107
7.13.3.10 UpdateTotalVerticeWeight()	107
7.13.4 Member Data Documentation	107
7.13.4.1 _value	108
7.13.4.2 edges	108
7.13.4.3 edgesV	108
7.13.4.4 total_edge_weights	108
7.14 Markov::Random::RandomEngine Class Reference	108
7.14.1 Detailed Description	109
7.14.2 Member Function Documentation	110
7.14.2.1 random()	110
7.15 Markov::API::CLI::Terminal Class Reference	110
7.15.1 Detailed Description	112
7.15.2 Member Enumeration Documentation	112
7.15.2.1 color	112
7.15.3 Constructor & Destructor Documentation	112
7.15.3.1 Terminal()	112

7.15.4 Member Data Documentation	112
7.15.4.1 colormap	113
7.15.4.2 endl	113
7.16 Markov::API::Concurrency::ThreadSharedListHandler Class Reference	113
7.16.1 Detailed Description	115
7.16.2 Constructor & Destructor Documentation	115
7.16.2.1 ThreadSharedListHandler()	115
7.16.3 Member Function Documentation	116
7.16.3.1 next()	116
7.16.4 Member Data Documentation	116
7.16.4.1 listfile	116
7.16.4.2 mlock	116
7.17 Markov::GUI::Train Class Reference	116
7.17.1 Detailed Description	118
7.17.2 Constructor & Destructor Documentation	118
7.17.2.1 Train()	118
7.17.3 Member Function Documentation	118
7.17.3.1 home	118
7.17.3.2 train	118
7.17.4 Member Data Documentation	118
7.17.4.1 ui	118
8 File Documentation	119
8.1 about.h File Reference	119
8.2 about.h	119
8.3 argparse.cpp File Reference	120
8.4 argparse.cpp	120
8.5 argparse.h File Reference	120
8.5.1 Macro Definition Documentation	122
8.5.1.1 BOOST_ALL_DYN_LINK	122
8.6 argparse.h	122
8.7 CLI.h File Reference	124
8.8 CLI.h	125
8.9 dllmain.cpp File Reference	125
8.10 dllmain.cpp	125
8.11 edge.h File Reference	126
8.12 edge.h	127
8.13 framework.h File Reference	128
8.13.1 Macro Definition Documentation	128
8.13.1.1 WIN32_LEAN_AND_MEAN	128
8.14 framework.h	128
8.15 main.cpp File Reference	129

8.15.1 Function Documentation	129
8.15.1.1 main()	129
8.16 src/main.cpp	130
8.17 main.cpp File Reference	130
8.17.1 Function Documentation	131
8.17.1.1 main()	131
8.18 UI/src/main.cpp	131
8.19 markopy.cpp File Reference	132
8.19.1 Macro Definition Documentation	133
8.19.1.1 BOOST_PYTHON_STATIC_LIB	133
8.20 markopy.cpp	133
8.21 markopy_cli.py File Reference	133
8.22 markopy_cli.py	134
8.23 markovPasswords.cpp File Reference	136
8.24 markovPasswords.cpp	136
8.25 markovPasswords.h File Reference	138
8.26 markovPasswords.h	139
8.27 MarkovPasswordsGUI.cpp File Reference	140
8.28 MarkovPasswordsGUI.cpp	140
8.29 MarkovPasswordsGUI.h File Reference	141
8.30 MarkovPasswordsGUI.h	142
8.31 menu.cpp File Reference	142
8.32 menu.cpp	142
8.33 menu.h File Reference	143
8.34 menu.h	144
8.35 model.h File Reference	144
8.36 model.h	145
8.37 model_2gram.py File Reference	148
8.38 model_2gram.py	148
8.39 node.h File Reference	148
8.40 node.h	149
8.41 pch.cpp File Reference	151
8.42 MarkovModel/src/pch.cpp	151
8.43 pch.cpp File Reference	151
8.44 UnitTests/pch.cpp	152
8.45 pch.h File Reference	152
8.46 MarkovModel/src/pch.h	153
8.47 pch.h File Reference	153
8.48 UnitTests/pch.h	153
8.49 random-model.py File Reference	154
8.50 random-model.py	154
8.51 random.h File Reference	154

8.52 random.h	156
8.53 README.md File Reference	157
8.54 term.cpp File Reference	157
8.54.1 Function Documentation	157
8.54.1.1 operator<<()	157
8.55 term.cpp	157
8.56 term.h File Reference	158
8.56.1 Macro Definition Documentation	160
8.56.1.1 TERM_FAIL	160
8.56.1.2 TERM_INFO	160
8.56.1.3 TERM_SUCC	160
8.56.1.4 TERM_WARN	160
8.57 term.h	160
8.58 threadSharedListHandler.cpp File Reference	161
8.59 threadSharedListHandler.cpp	161
8.60 threadSharedListHandler.h File Reference	161
8.61 threadSharedListHandler.h	162
8.62 Train.h File Reference	163
8.63 Train.h	163
8.64 UnitTests.cpp File Reference	164
8.65 UnitTests.cpp	165
Index	173

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

- C++ 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

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

markopy_cli	15
Markov	
Namespace for the markov-model related classes. Contains Model , Node and Edge classes	18
Markov::API	
Namespace for the MarkovPasswords API	19
Markov::API::CLI	
Structure to hold parsed cli arguments	19
Markov::API::Concurrency	
Namespace for Concurrency related classes	20
Markov::GUI	
Namespace for MarkovPasswords API GUI wrapper	20
Markov::Markopy	21
Markov::Random	
Objects related to RNG	21
model_2gram	22
random	22
random-model	23
Testing	
Namespace for Microsoft Native Unit Testing Classes	23
Testing::MarkovModel	
Testing namespace for MarkovModel	23
Testing::MarkovPasswords	
Testing namespace for MarkovPasswords	26
Testing::MVP	
Testing Namespace for Minimal Viable Product	26
Testing::MVP::MarkovModel	
Testing Namespace for MVP MarkovModel	27
Testing::MVP::MarkovPasswords	
Testing namespace for MVP MarkovPasswords	33

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Markov::API::CLI::_programOptions	37
Markov::API::CLI::Argparse	42
Markov::Edge< NodeStorageType >	53
Markov::Edge< char >	53
Markov::Edge< storageType >	53
Markov::Model< NodeStorageType >	90
Markov::Model< char >	90
Markov::API::MarkovPasswords	61
Markov::Node< storageType >	99
Markov::Node< char >	99
Markov::Node< NodeStorageType >	99
QMainWindow	
Markov::GUI::about	40
Markov::GUI::CLI	47
Markov::GUI::MarkovPasswordsGUI	75
Markov::GUI::menu	84
Markov::GUI::Train	116
Markov::Random::RandomEngine	108
Markov::Random::DefaultRandomEngine	49
Markov::Random::Marsaglia	78
Markov::Random::Mersenne	86
Markov::API::CLI::Terminal	110
Markov::API::Concurrency::ThreadSharedListHandler	113

Chapter 4

Class Index

4.1 Class List

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

Markov::API::CLI::_programOptions	
Structure to hold parsed cli arguments	37
Markov::GUI::about	
QT Class for about page	40
Markov::API::CLI::Argparse	
Parse command line arguments	42
Markov::GUI::CLI	
QT CLI Class	47
Markov::Random::DefaultRandomEngine	
Implementation using Random.h default random engine	49
Markov::Edge< NodeStorageType >	
Edge class used to link nodes in the model together	53
Markov::API::MarkovPasswords	
Markov::Model with char represented nodes	61
Markov::GUI::MarkovPasswordsGUI	
Reporting UI	75
Markov::Random::Marsaglia	
Implementation of Marsaglia Random Engine	78
Markov::GUI::menu	
QT Menu class	84
Markov::Random::Mersenne	
Implementation of Mersenne Twister Engine	86
Markov::Model< NodeStorageType >	
Class for the final Markov Model , constructed from nodes and edges	90
Markov::Node< storageType >	
A node class that for the vertices of model. Connected with eachother using Edge	99
Markov::Random::RandomEngine	
An abstract class for Random Engine	108
Markov::API::CLI::Terminal	
Pretty colors for Terminal . Windows Only	110
Markov::API::Concurrency::ThreadSharedListHandler	
Simple class for managing shared access to file	113
Markov::GUI::Train	
QT Training page class	116

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

about.h	119
argparse.cpp	120
argparse.h	120
CLI.h	124
dllmain.cpp	125
edge.h	126
framework.h	128
src/main.cpp	129
UI/src/main.cpp	130
markopy.cpp	132
markopy_cli.py	133
markovPasswords.cpp	136
markovPasswords.h	138
MarkovPasswordsGUI.cpp	140
MarkovPasswordsGUI.h	141
menu.cpp	142
menu.h	143
model.h	144
model_2gram.py	148
node.h	148
MarkovModel/src/pch.cpp	151
UnitTests/pch.cpp	151
MarkovModel/src/pch.h	152
UnitTests/pch.h	153
random-model.py	154
random.h	154
term.cpp	157
term.h	158
threadSharedListHandler.cpp	161
threadSharedListHandler.h	161
Train.h	163
UnitTests.cpp	164

Chapter 6

Namespace Documentation

6.1 markopy_cli Namespace Reference

Functions

- def `cli_init` (input_model)
- def `cli_train` (model, dataset, separator, 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`

6.1.1 Detailed Description

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

6.1.2 Function Documentation

6.1.2.1 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
```

6.1.2.2 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
```

6.1.2.3 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
```

6.1.3 Variable Documentation

6.1.3.1 action

markopy_cli.action

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

6.1.3.2 args

markopy_cli.args = parser.parse_args()

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

6.1.3.3 bulk

markopy_cli.bulk

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

6.1.3.4 corpus_list

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

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

6.1.3.5 default

markopy_cli.default

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

6.1.3.6 help

markopy_cli.help

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

6.1.3.7 model

def markopy_cli.model = cli_init(args.input)

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

6.1.3.8 model_base

markopy_cli.model_base = input

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

6.1.3.9 model_extension

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

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

6.1.3.10 model_list

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

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

6.1.3.11 output

```
markopy_cli.output
```

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

6.1.3.12 output_file_name

```
markopy_cli.output_file_name = corpus
```

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

6.1.3.13 output_forced

```
markopy_cli.output_forced
```

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

6.1.3.14 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](#).

6.1.3.15 True

```
markopy_cli.True
```

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

6.2 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](#).

6.2.1 Detailed Description

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

6.3 Markov::API Namespace Reference

Namespace for the [MarkovPasswords API](#).

Namespaces

- [CLI](#)
Structure to hold parsed cli arguments.
- [Concurrency](#)
Namespace for [Concurrency](#) related classes.

Classes

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

6.3.1 Detailed Description

Namespace for the [MarkovPasswords API](#).

6.4 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)

6.4.1 Detailed Description

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

6.4.2 Typedef Documentation

6.4.2.1 ProgramOptions

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

6.4.3 Function Documentation

6.4.3.1 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](#).

6.5 Markov::API::Concurrency Namespace Reference

Namespace for [Concurrency](#) related classes.

Classes

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

6.5.1 Detailed Description

Namespace for [Concurrency](#) related classes.

6.6 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.

6.6.1 Detailed Description

namespace for MarkovPasswords [API GUI](#) wrapper

6.7 Markov::Markopy Namespace Reference

Functions

- [BOOST_PYTHON_MODULE](#) (markopy)

6.7.1 Function Documentation

6.7.1.1 BOOST_PYTHON_MODULE()

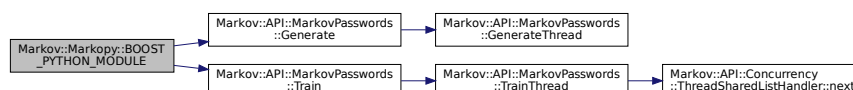
Markov::Markopy::BOOST_PYTHON_MODULE (
markopy)

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::API::MarkovPasswords::Generate\(\)](#), and [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the call graph for this function:



6.8 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](#).

6.8.1 Detailed Description

Objects related to RNG.

6.9 model_2gram Namespace Reference

Variables

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

6.9.1 Detailed Description

python script for generating a 2gram model

6.9.2 Variable Documentation

6.9.2.1 alphabet

model_2gram.alphabet = string.printable

password alphabet

Definition at line 10 of file [model_2gram.py](#).

6.9.2.2 f

model_2gram.f = open('../models/2gram.mdl', "wb")

output file handle

Definition at line 16 of file [model_2gram.py](#).

Referenced by [Markov::Model< char >.Export\(\)](#), and [Markov::Model< char >.Import\(\)](#).

6.10 random Namespace Reference

6.10.1 Detailed Description

-model

python script for generating a 2gram model

6.11 random-model Namespace Reference

Variables

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

6.11.1 Variable Documentation

6.11.1.1 alphabet

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

6.11.1.2 f

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

6.12 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.

6.12.1 Detailed Description

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

6.13 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.

6.13.1 Detailed Description

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

6.13.2 Function Documentation

6.13.2.1 TEST_CLASS() [1/3]

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

Test class for rest of Edge cases.

send exception on integer underflow

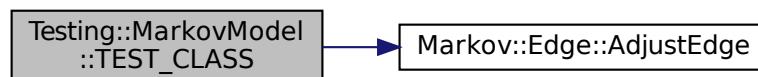
test integer overflows

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,
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         }
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);
00519                 e->AdjustEdge(~0ull);
00520                 e->AdjustEdge(1);
00521                 delete LeftNode;
00522                 delete RightNode;
00523                 delete e;
00524             };
00525             Assert::ExpectException<std::underflow_error>(_overflow_adjust);
00526         }
00527     };
```

References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#).

Here is the call graph for this function:



6.13.2.2 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:
```

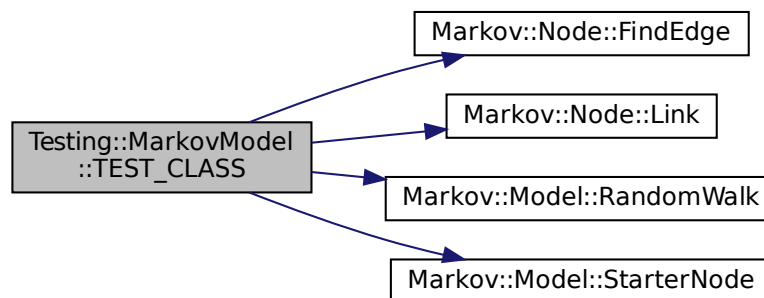
```

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 [Markov::Node< storageType >::FindEdge\(\)](#), [Markov::Node< storageType >::Link\(\)](#), [Markov::Model< NodeStorageType >::RandomWalk\(\)](#) and [Markov::Model< NodeStorageType >::StarterNode\(\)](#).

Here is the call graph for this function:



6.13.2.3 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');

```

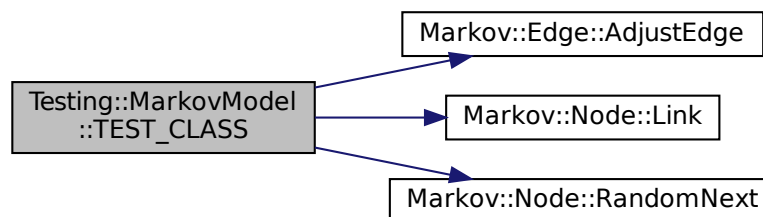
```

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 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 [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Link\(\)](#), and [Markov::Node< storageType >::RandomNext\(\)](#).

Here is the call graph for this function:



6.14 Testing::MarkovPasswords Namespace Reference

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

6.14.1 Detailed Description

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

6.15 Testing::MVP Namespace Reference

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

Namespaces

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

6.15.1 Detailed Description

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

6.16 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.

6.16.1 Detailed Description

[Testing](#) Namespace for MVP MarkovModel.

6.16.2 Function Documentation

6.16.2.1 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,
00040                                     RightNode);
00041             Assert::IsTrue(LeftNode == e->LeftNode());
00042             Assert::IsTrue(RightNode == e->RightNode());
00043             delete LeftNode;
00043             delete RightNode;
```

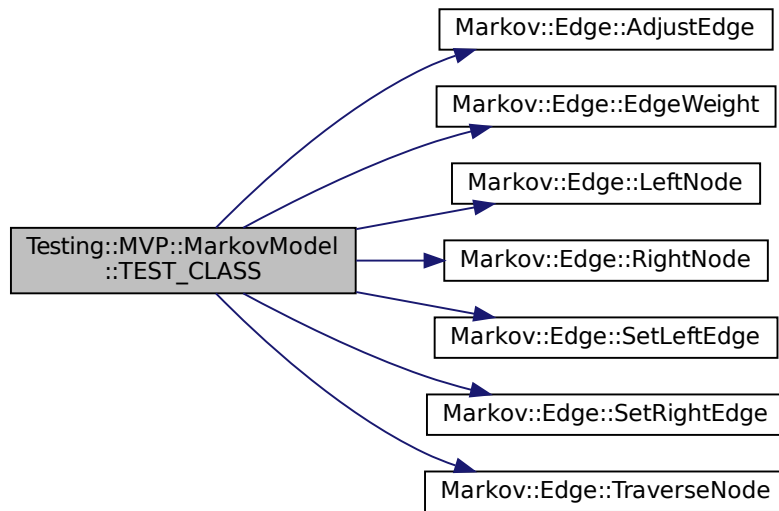
```

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
00076     TEST_METHOD(set_left_and_right) {
00077         Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00078         Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00079         Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(LeftNode,
RightNode);
00080
00081         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
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 };

```

References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Edge< NodeStorageType >::LeftNode\(\)](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Edge< NodeStorageType >::SetLeftEdge\(\)](#), [Markov::Edge< NodeStorageType >::SetRightEdge\(\)](#), and [Markov::Edge< NodeStorageType >::TraverseNode\(\)](#).

Here is the call graph for this function:



6.16.2.2 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

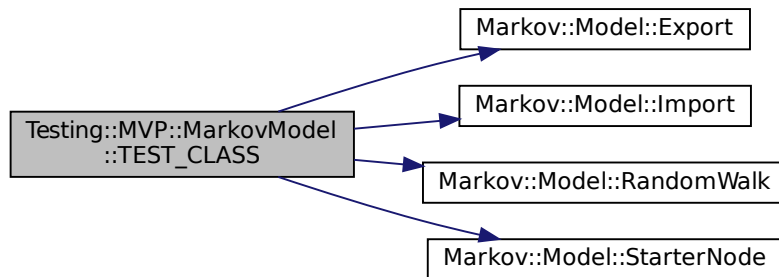
Definition at line 347 of file [UnitTests.cpp](#).

```

00348     {
00349     public:
00352         TEST_METHOD(model_constructor) {
00353             Markov::Model<unsigned char> m;
00354             Assert::AreEqual((unsigned char)'0', m.StarterNode()->NodeValue());
00355         }
00356
00359         TEST_METHOD(import_filename) {
00360             Markov::Model<unsigned char> m;
00361             Assert::IsTrue(m.Import("../MarkovPasswords/Models/2gram.mdl"));
00362         }
00363
00366         TEST_METHOD(export_filename) {
00367             Markov::Model<unsigned char> m;
00368             Assert::IsTrue(m.Export("../MarkovPasswords/Models/testcase.mdl"));
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 [Markov::Model< NodeStorageType >::Export\(\)](#), [Markov::Model< NodeStorageType >::Import\(\)](#), [Markov::Model< NodeStorageType >::RandomWalk\(\)](#), and [Markov::Model< NodeStorageType >::StarterNode\(\)](#).

Here is the call graph for this function:



6.16.2.3 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
00162     TEST_METHOD(rand_next_low) {
00163
00164         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00165         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00166         Markov::Edge<unsigned char>* e = src->Link(target1);
00167         e->AdjustEdge(15);
00168         Markov::Node<unsigned char>* res = src->RandomNext();
00169         Assert::IsTrue(res == target1);
00170         delete src;
00171         delete target1;
00172         delete e;
00173     }
00174
00175     TEST_METHOD(rand_next_u32) {
00176
00177         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00178         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00179         Markov::Edge<unsigned char>* e = src->Link(target1);
00180         e->AdjustEdge(1 << 31);
00181         Markov::Node<unsigned char>* res = src->RandomNext();
00182         Assert::IsTrue(res == target1);
00183         delete src;
00184         delete target1;
00185         delete e;
00186     }
00187
00188     TEST_METHOD(rand_next_choice_1) {
00189
00190         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00191         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00192         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00193         Markov::Edge<unsigned char>* e1 = src->Link(target1);
00194         Markov::Edge<unsigned char>* e2 = src->Link(target2);
00195         e1->AdjustEdge(1);
00196         e2->AdjustEdge((unsigned long)(1ull << 31));
00197         Markov::Node<unsigned char>* res = src->RandomNext();
00198         Assert::IsNotNull(res);
00199         Assert::IsTrue(res == target2);
00200         delete src;
00201         delete target1;
00202         delete e1;
00203         delete e2;
00204     }
00205
00206     TEST_METHOD(rand_next_choice_2) {
00207
00208         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00209         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00210         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00211         Markov::Edge<unsigned char>* e1 = src->Link(target1);
00212         Markov::Edge<unsigned char>* e2 = src->Link(target2);
00213         e1->AdjustEdge(1);
00214         e2->AdjustEdge((unsigned long)(1ull << 31));
00215         Markov::Node<unsigned char>* res = src->RandomNext();
00216         Assert::IsNotNull(res);
00217         Assert::IsTrue(res == target1);
00218         delete src;
00219         delete target1;
00220         delete e1;
00221         delete e2;
00222     }
00223
00224     TEST_METHOD(update_edges_count) {
00225
00226         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00227         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00228         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00229         Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00230         Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00231         e1->AdjustEdge(25);
00232         src->UpdateEdges(e1);
00233         e2->AdjustEdge(30);

```

```

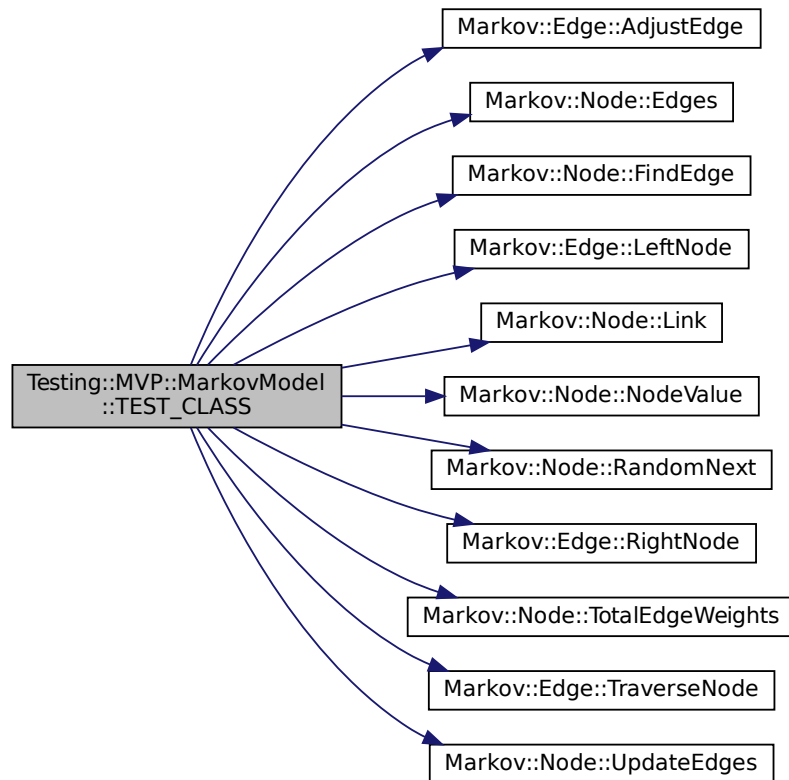
00247         src->UpdateEdges(e2);
00248
00249         Assert::AreEqual((size_t)2, src->Edges()->size());
00250
00251         delete src;
00252         delete target1;
00253         delete e1;
00254         delete e2;
00255
00256     }
00257
00260     TEST_METHOD(update_edges_total) {
00261
00262         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 [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Node< storageType >::FindEdge\(\)](#), [Markov::Edge< NodeStorageType >::LeftNode\(\)](#), [Markov::Node< storageType >::Link](#), [Markov::Node< storageType >::NodeValue\(\)](#), [Markov::Node< storageType >::RandomNext\(\)](#), [Markov::Edge< NodeStorageType >::TotalEdgeWeights\(\)](#), [Markov::Edge< NodeStorageType >::TraverseNode\(\)](#), and [Markov::Node< storageType >::UpdateEdges\(\)](#).

Here is the call graph for this function:



6.17 Testing::MVP::MarkovPasswords Namespace Reference

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

Functions

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

6.17.1 Detailed Description

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

6.17.2 Function Documentation

6.17.2.1 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     };
```

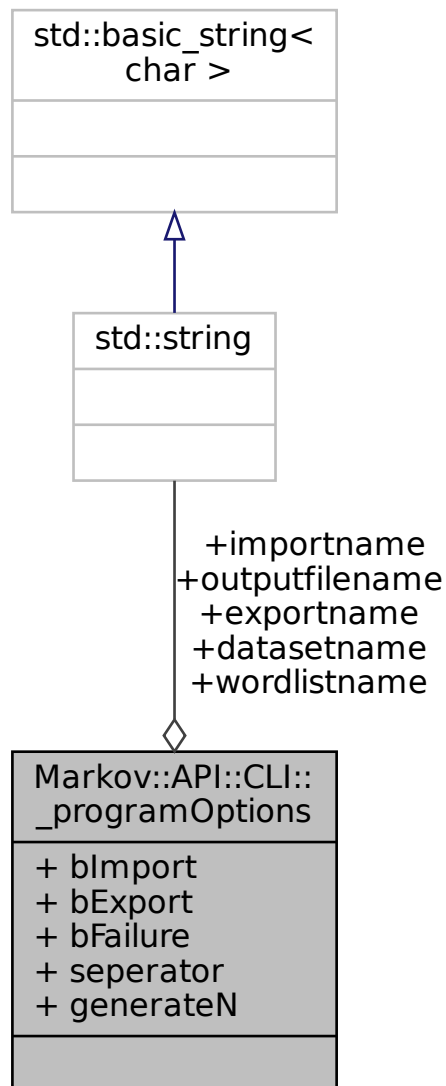

Chapter 7

Class Documentation

7.1 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`

7.1.1 Detailed Description

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

7.1.2 Member Data Documentation

7.1.2.1 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\(\)](#).

7.1.2.2 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\(\)](#).

7.1.2.3 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\(\)](#).

7.1.2.4 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\(\)](#).

7.1.2.5 exportname

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

7.1.2.6 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\(\)](#).

7.1.2.7 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\(\)](#).

7.1.2.8 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\(\)](#).

7.1.2.9 seperator

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

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

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

7.1.2.10 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](#)

7.2 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](#)

7.2.1 Detailed Description

QT Class for about page.

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

7.2.2 Constructor & Destructor Documentation

7.2.2.1 about()

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

7.2.3 Member Data Documentation

7.2.3.1 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:

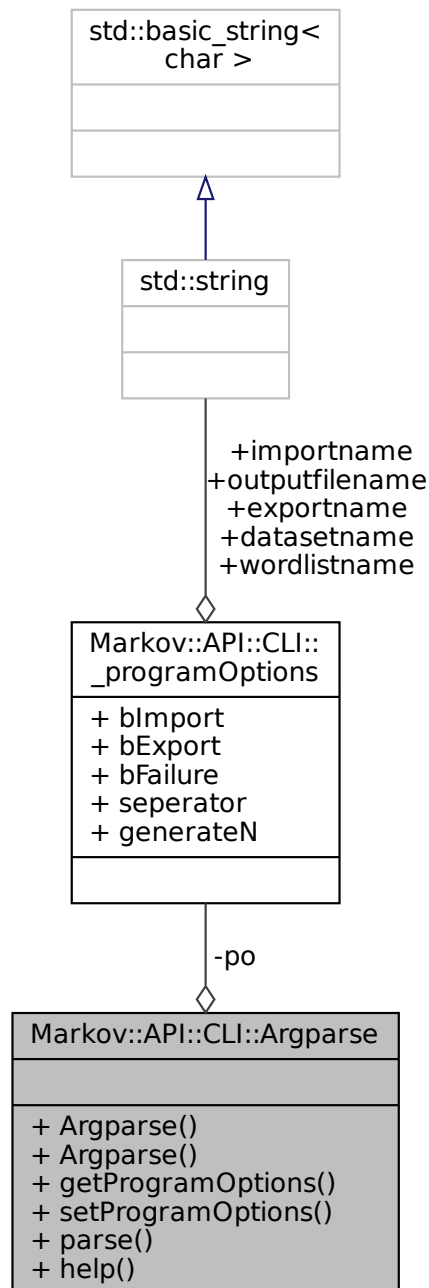
- [about.h](#)

7.3 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`

7.3.1 Detailed Description

Parse command line arguments.

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

7.3.2 Constructor & Destructor Documentation

7.3.2.1 Argparse() [1/2]

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

7.3.2.2 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         if (vm.count("dataset") == 0) this->po.datasetname = "NULL";
00097         else if (vm.count("dataset") == 1) {
00098             this->po.datasetname = vm["dataset"].as<std::string>();
00099         }
00100         else {
00101             this->po.bFailure = true;
00102             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00103             std::cout << desc << std::endl;
00104         }
00105
00106         if (vm.count("wordlist") == 0) this->po.wordlistname = "NULL";
00107         else if (vm.count("wordlist") == 1) {
00108             this->po.wordlistname = vm["wordlist"].as<std::string>();
00109         }
00110         else {
00111             this->po.bFailure = true;
00112             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00113             std::cout << desc << std::endl;
00114         }
00115
00116         if (vm.count("import") == 0) this->po.importname = "NULL";
00117         else if (vm.count("import") == 1) {
00118             this->po.importname = vm["import"].as<std::string>();
00119             this->po.bImport = true;
00120         }
00121         else {
00122             this->po.bFailure = true;
00123             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00124             std::cout << desc << std::endl;
00125         }
00126
00127         if (vm.count("count") == 0) this->po.generateN = 0;
00128         else if (vm.count("count") == 1) {
00129             this->po.generateN = vm["count"].as<int>();
00130         }
00131         else {
00132             this->po.bFailure = true;
00133             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00134             std::cout << desc << std::endl;
00135         }
00136
00137         /*std::cout << vm["output"].as<std::string>() << std::endl;
std::cout << vm["dataset"].as<std::string>() << std::endl;
std::cout << vm["wordlist"].as<std::string>() << std::endl;
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](#).

7.3.3 Member Function Documentation

7.3.3.1 getProgramOptions()

```

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

```

Getter for command line options.

Getter for ProgramOptions populated by the arguement parser

Returns

ProgramOptions structure.

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

```

00155
00156         return this->po;
00157     }

```

References [po](#).

7.3.3.2 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
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 }

```

7.3.3.3 parse()

```

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

```

parse cli commands and return

Parameters

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

Returns

ProgramOptions structure.

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

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

7.3.3.4 setProgramOptions()

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

Initialize program options structure.

Parameters

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

Definition at line 172 of file [argparse.h](#).

```
00172
00173         {
00174             this->po.bImport = i;
00175             this->po.bExport = e;
00176             this->po.seperator = s;
00177             this->po.bFailure = bf;
00178             this->po.generateN = n;
00179             this->po.importname = iName;
00180             this->po.exportname = exName;
00181             this->po.outputfilename = oName;
00182             this->po.datasetname = dName;
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](#).

7.3.4 Member Data Documentation

7.3.4.1 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:

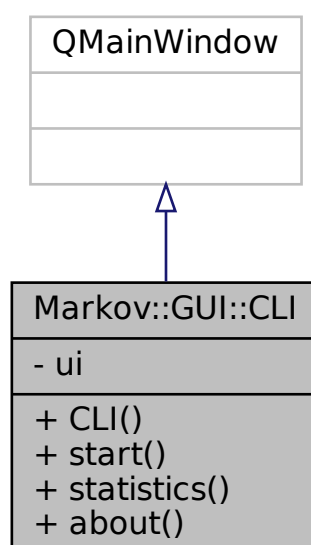
- [argparse.h](#)
- [argparse.cpp](#)

7.4 Markov::GUI::CLI Class Reference

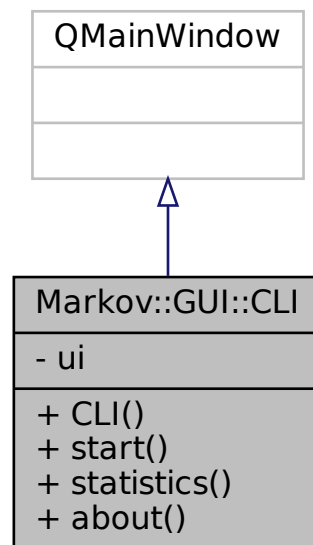
QT [CLI](#) Class.

```
#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](#)

7.4.1 Detailed Description

QT [CLI](#) Class.

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

7.4.2 Constructor & Destructor Documentation

7.4.2.1 CLI()

```

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

7.4.3 Member Function Documentation

7.4.3.1 about

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

7.4.3.2 start

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

7.4.3.3 statistics

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

7.4.4 Member Data Documentation

7.4.4.1 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:

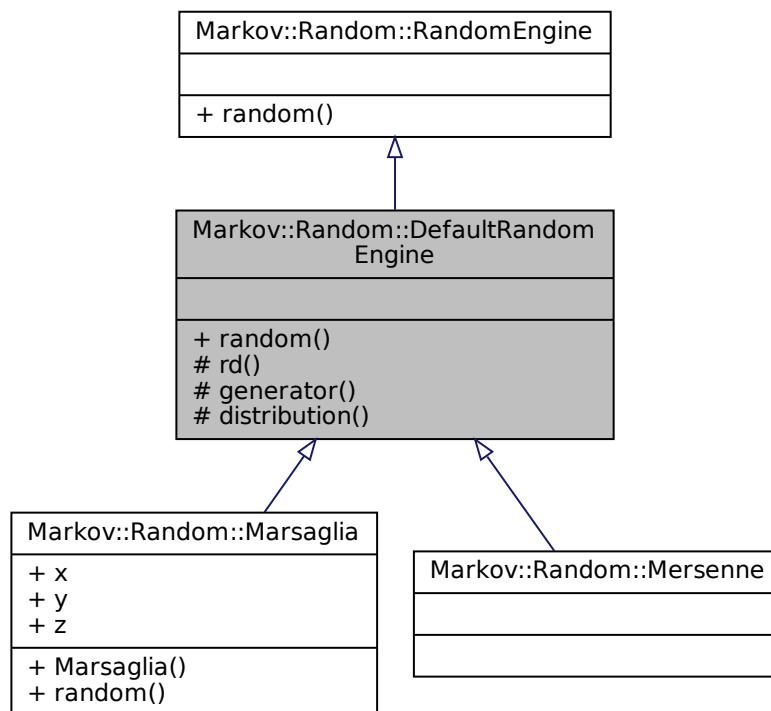
- [CLI.h](#)

7.5 Markov::Random::DefaultRandomEngine Class Reference

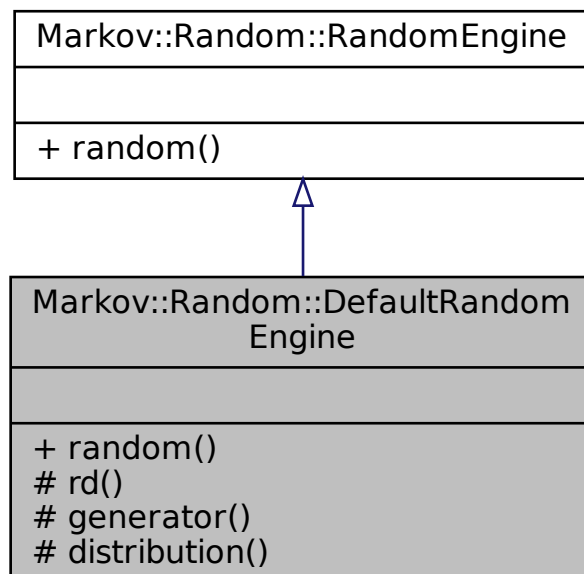
Implementation using [Random.h](#) default random engine.

```
#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.

7.5.1 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](#).

7.5.2 Member Function Documentation

7.5.2.1 distribution()

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

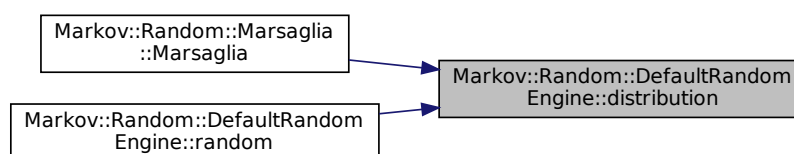
Distribution schema for seeding.

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:



7.5.2.2 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     }
```

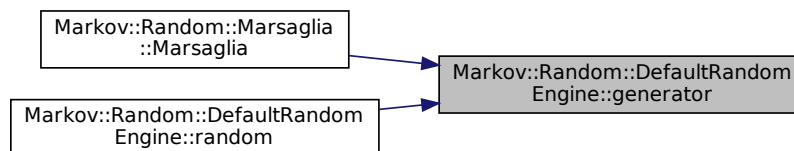
References [rd\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.2.3 random()

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

Generate [Random](#) Number.

Returns

random number in long range.

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

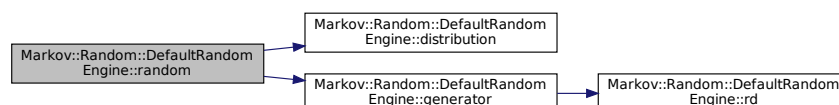
Reimplemented in [Markov::Random::Marsaglia](#).

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:



7.5.2.4 rd()

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

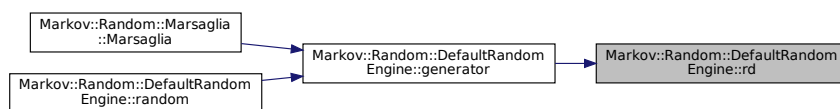
Default random device for seeding.

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:

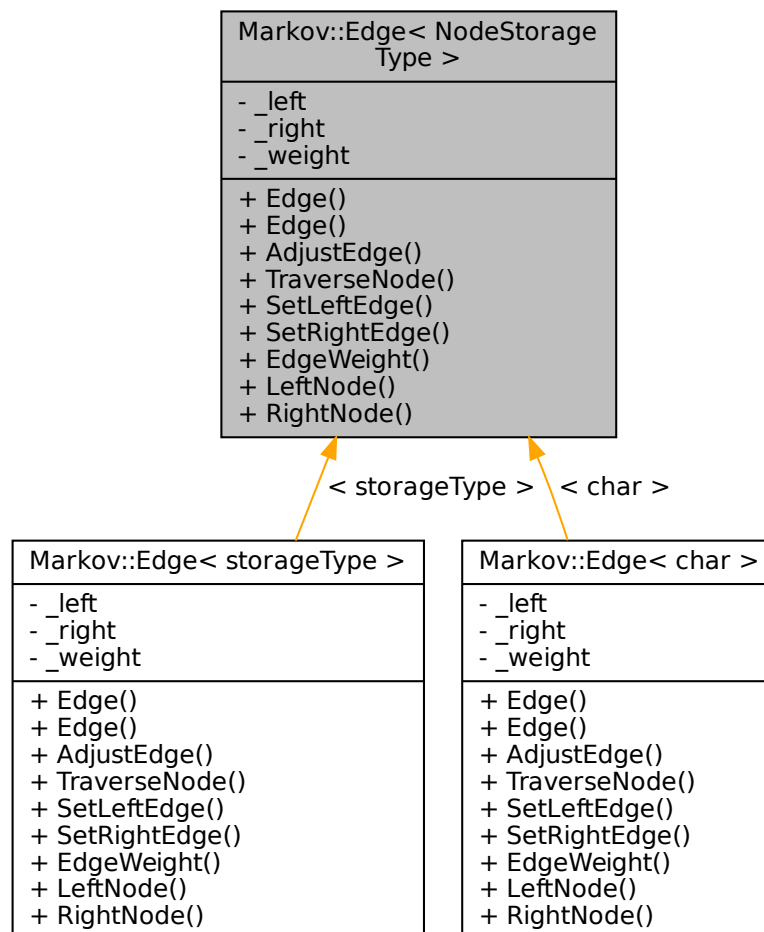
- [random.h](#)

7.6 Markov::Edge< NodeStorageType > Class Template Reference

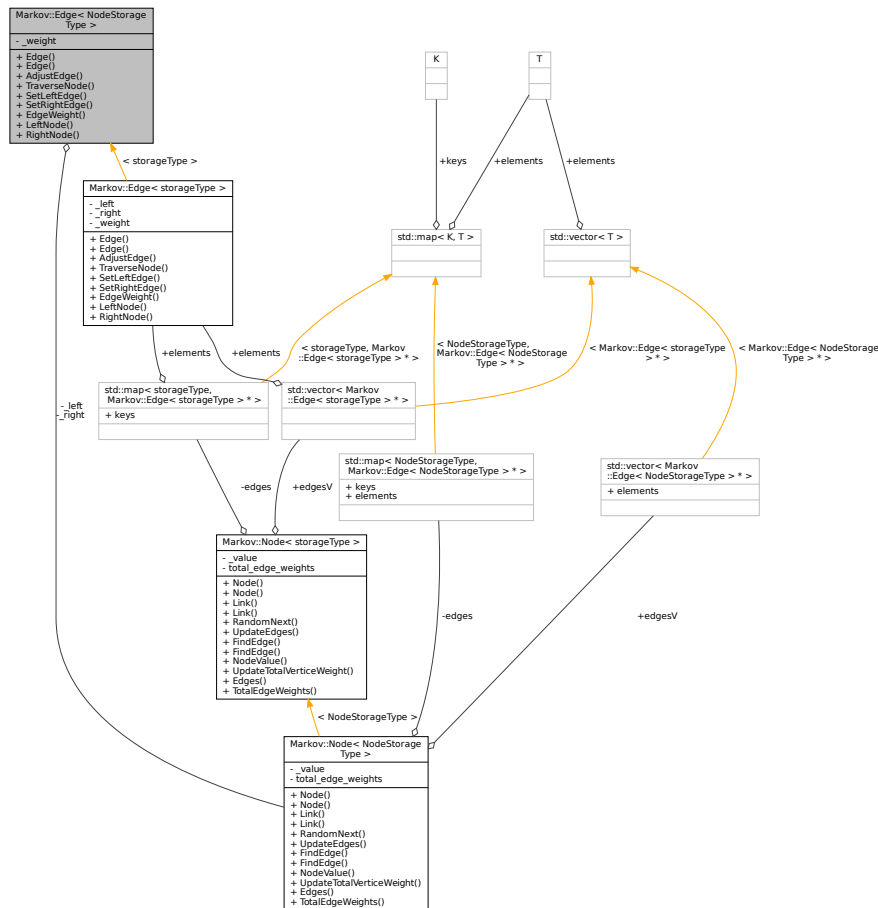
[Edge](#) class used to link nodes in the model together.

```
#include <edge.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

7.6.1 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 16 of file [edge.h](#).

7.6.2 Constructor & Destructor Documentation

7.6.2.1 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     }
```

7.6.2.2 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     }
```

7.6.3 Member Function Documentation

7.6.3.1 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

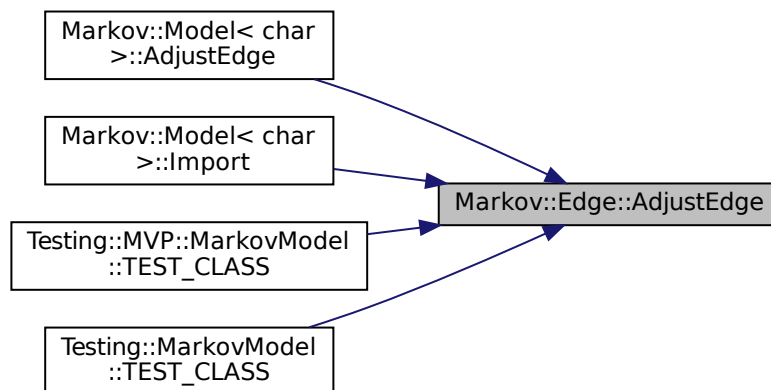
```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
e1->AdjustEdge(25);
```

Definition at line 119 of file [edge.h](#).

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

Referenced by [Markov::Model< char >::AdjustEdge\(\)](#), [Markov::Model< char >::Import\(\)](#), [Testing::MVP::MarkovModel::TEST_CLASS](#) and [Testing::MarkovModel::TEST_CLASS](#).

Here is the caller graph for this function:



7.6.3.2 EdgeWeight()

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

Returns

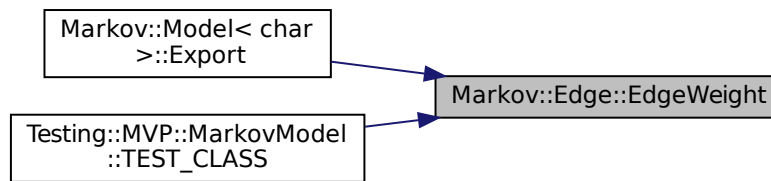
edge's EdgeWeight.

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

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

Referenced by [Markov::Model< char >::Export\(\)](#), and [Testing::MVP::MarkovModel::TEST_CLASS](#).

Here is the caller graph for this function:



7.6.3.3 LeftNode()

```

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

Returns

edge's LeftNode.

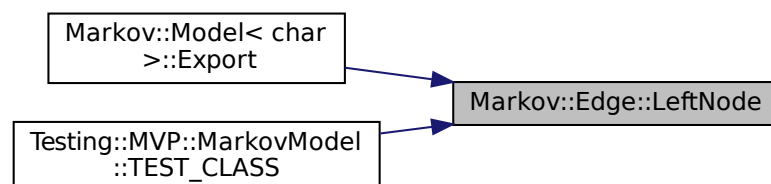
Definition at line 147 of file [edge.h](#).

```

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

Referenced by [Markov::Model< char >::Export\(\)](#), and [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#).

Here is the caller graph for this function:



7.6.3.4 RightNode()

```

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

Returns

edge's RightNode.

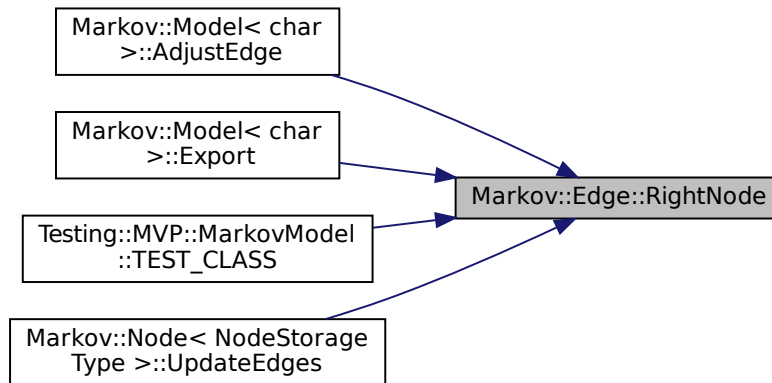
Definition at line 152 of file [edge.h](#).

```

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

Referenced by [Markov::Model< char >::AdjustEdge\(\)](#), [Markov::Model< char >::Export\(\)](#), [Testing::MVP::MarkovModel::TEST_CLASS](#) and [Markov::Node< NodeStorageType >::UpdateEdges\(\)](#).

Here is the caller graph for this function:



7.6.3.5 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.
-------------	---

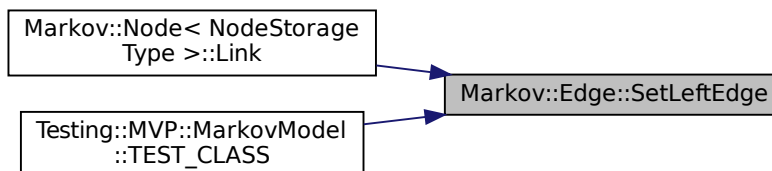
Definition at line 132 of file [edge.h](#).

```

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

Referenced by [Markov::Node< NodeStorageType >::Link\(\)](#), and [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#).

Here is the caller graph for this function:



7.6.3.6 SetRightEdge()

```

template<typename NodeStorageType >
  
```

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

Parameters

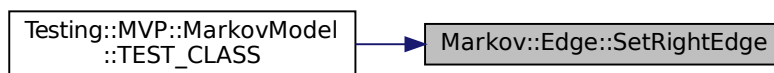
<i>node</i>	- Node to be linked with.
-------------	---

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

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

Referenced by [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#).

Here is the caller graph for this function:



7.6.3.7 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

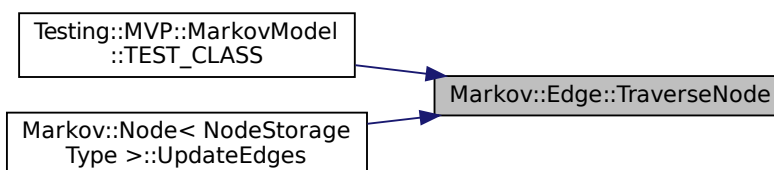
```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
e1->AdjustEdge(25);
Markov::Edge<unsigned char>* e2 = e1->traverseNode();
```

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

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

Referenced by [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#), and [Markov::Node< NodeStorageType >::UpdateEdges\(\)](#).

Here is the caller graph for this function:



7.6.4 Member Data Documentation

7.6.4.1 `_left`

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

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

7.6.4.2 `_right`

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

source node

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

7.6.4.3 `_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 file:

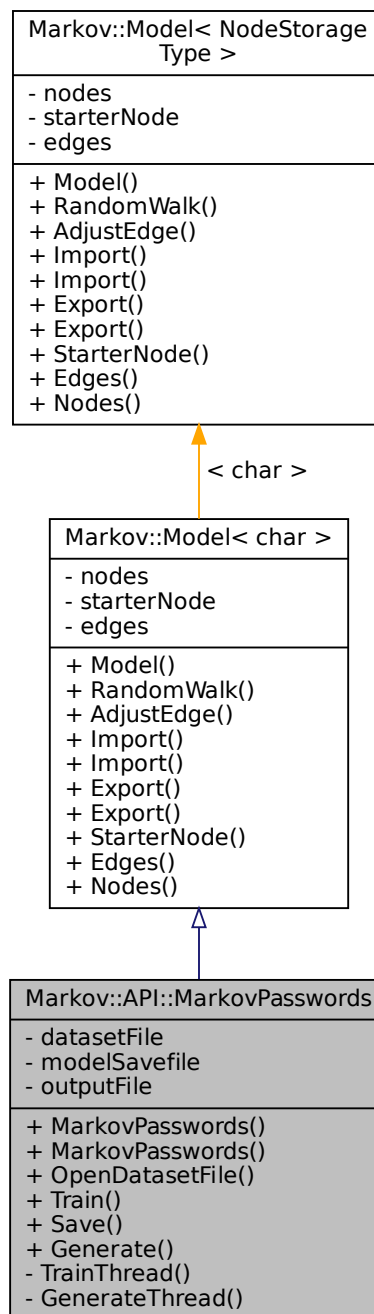
- [edge.h](#)

7.7 Markov::API::MarkovPasswords Class Reference

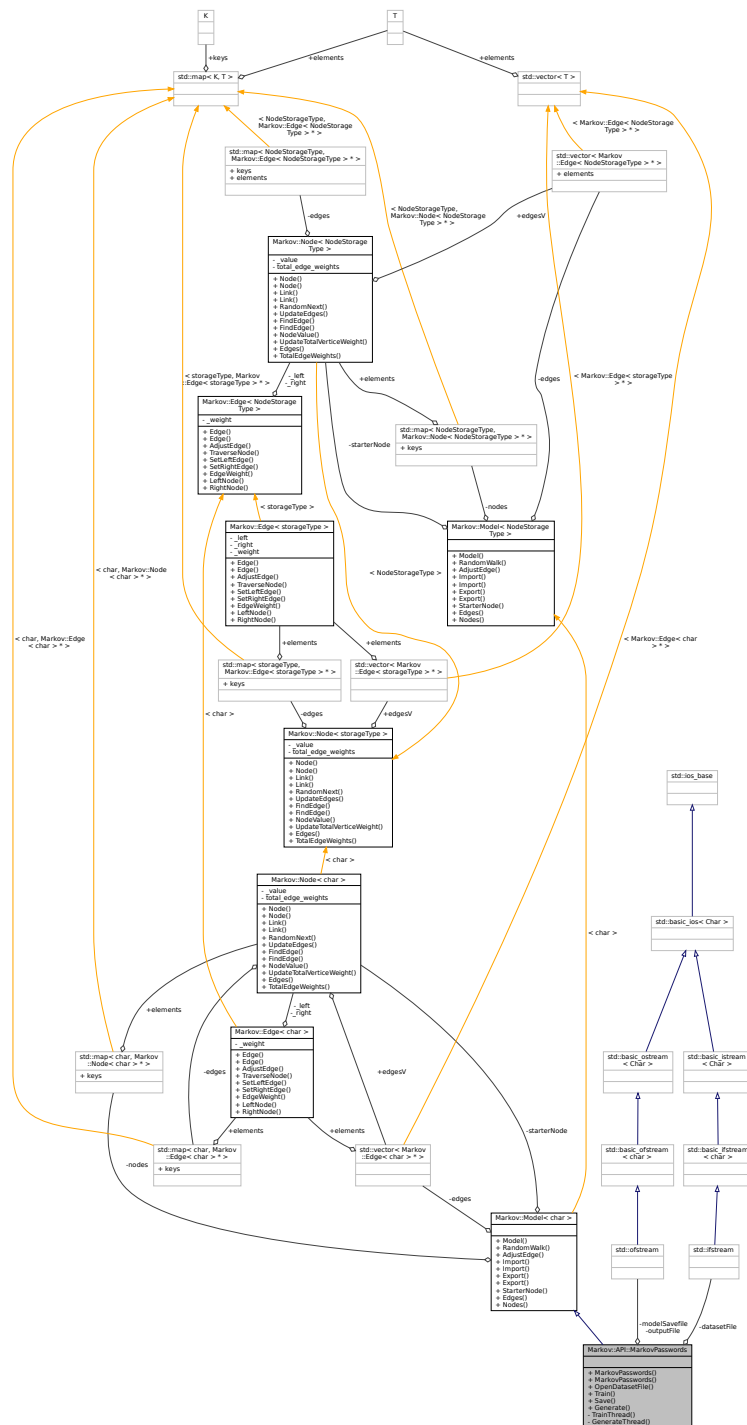
[Markov::Model](#) with char represented nodes.

```
#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.

7.7.1 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](#).

7.7.2 Constructor & Destructor Documentation

7.7.2.1 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 }
```

7.7.2.2 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 }
```

7.7.3 Member Function Documentation

7.7.3.1 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
---------------	--

Parameters

<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 }
```

7.7.3.2 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; }
```

7.7.3.3 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 }
```

7.7.3.4 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() <<
00280         "\n";
00281     }
00282     return true;
```

7.7.3.5 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.

Write the data to this->outputfile

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

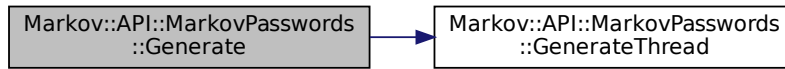
Definition at line 92 of file [markovPasswords.cpp](#).

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



7.7.3.6 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.

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

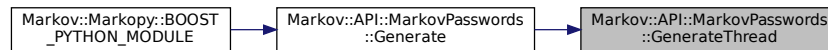
Definition at line 114 of file [markovPasswords.cpp](#).

```

00114
00115     char* res = new char[maxLen+5];
00116     if(n==0) return;
00117
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 }
  
```

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

Here is the caller graph for this function:



7.7.3.7 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 }
```

7.7.3.8 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         if (this->nodes.find(target) == this->nodes.end()) {
00229             targetN = new Markov::Node<NodeStorageType>(target);
00230             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00231             //std::cout << "Creating new node at end.\n";
00232         }
00233         e = new Markov::Edge<NodeStorageType>(srcN, targetN, oc);
00234         this->edges.insert(std::pair<Markov::Node<NodeStorageType>*, Markov::Edge<NodeStorageType>*>(srcN, e));
00235     }
00236 }
```

```

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 }

```

7.7.3.9 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; }
```

7.7.3.10 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 }

```

7.7.3.11 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     {
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     }
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 }

```

7.7.3.12 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 }

```

7.7.3.13 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; }

```

7.7.3.14 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

```

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

```

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

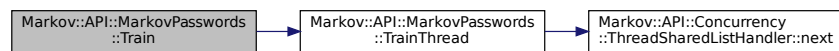
```

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 }
```

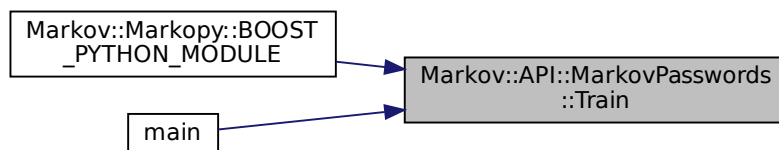
References [TrainThread\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.3.15 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     {
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);
```

```

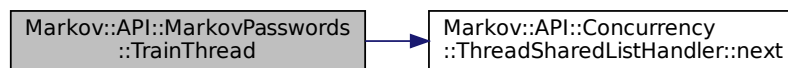
00067     }
00068     char* linebuf = new char[line.length()+5];
00069 #ifdef _WIN32
00070     sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
00071 #else
00072     sscanf(line.c_str(), format_str, &oc, linebuf);
00073 #endif
00074     this->AdjustEdge((const char*)linebuf, oc);
00075     delete linebuf;
00076 }
00077 }

```

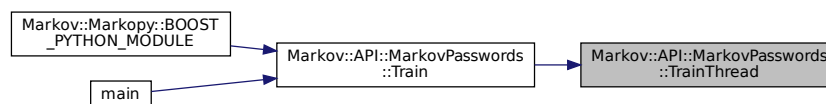
References [Markov::API::Concurrency::ThreadSharedListHandler::next\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.4 Member Data Documentation

7.7.4.1 datasetFile

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

Definition at line 100 of file [markovPasswords.h](#).

7.7.4.2 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](#).

7.7.4.3 modelSavefile

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

Definition at line 101 of file [markovPasswords.h](#).

7.7.4.4 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](#).

7.7.4.5 outputFile

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

Definition at line 102 of file [markovPasswords.h](#).

7.7.4.6 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:

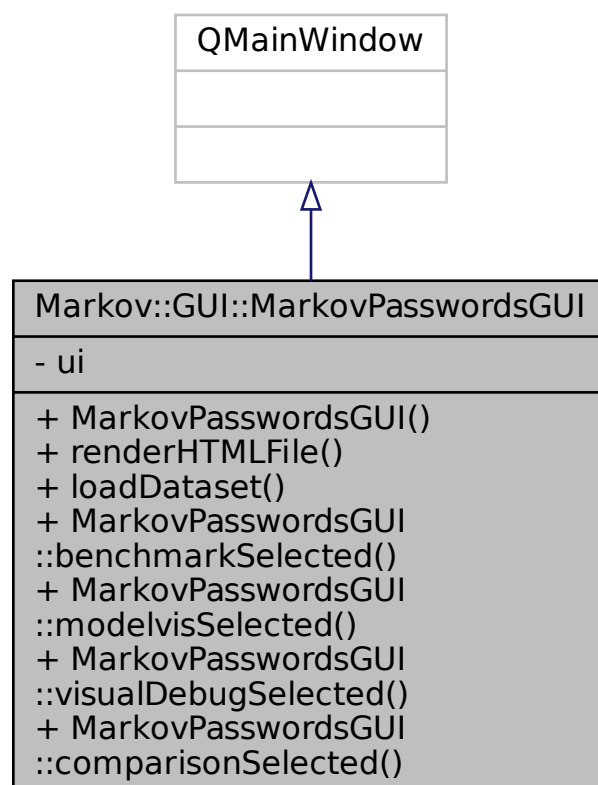
- [markovPasswords.h](#)
- [markovPasswords.cpp](#)

7.8 Markov::GUI::MarkovPasswordsGUI Class Reference

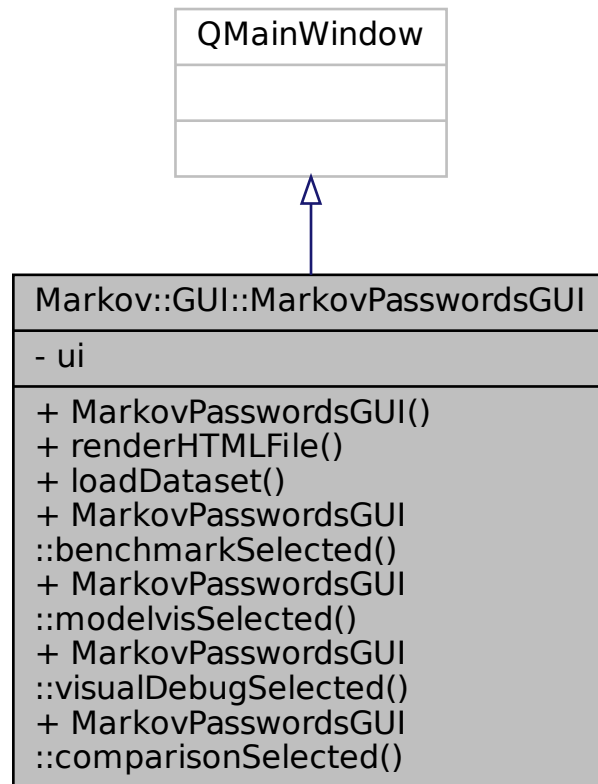
Reporting UI.

`#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](#)

7.8.1 Detailed Description

Reporting UI.

UI for reporting and debugging tools for MarkovPassword

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

7.8.2 Constructor & Destructor Documentation

7.8.2.1 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 }
```

References [ui](#).

7.8.3 Member Function Documentation

7.8.3.1 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 }
```

7.8.3.2 MarkovPasswordsGUI::benchmarkSelected

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

7.8.3.3 MarkovPasswordsGUI::comparisonSelected

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

7.8.3.4 MarkovPasswordsGUI::modelvisSelected

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

7.8.3.5 MarkovPasswordsGUI::visualDebugSelected

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

7.8.3.6 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 }
```

7.8.4 Member Data Documentation

7.8.4.1 ui

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

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

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

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

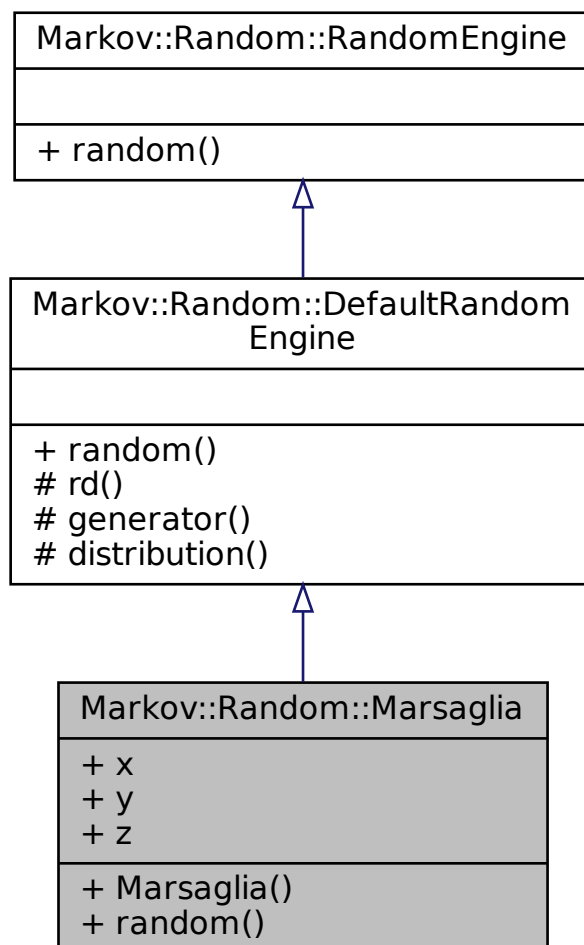
- [MarkovPasswordsGUI.h](#)
- [MarkovPasswordsGUI.cpp](#)

7.9 Markov::Random::Marsaglia Class Reference

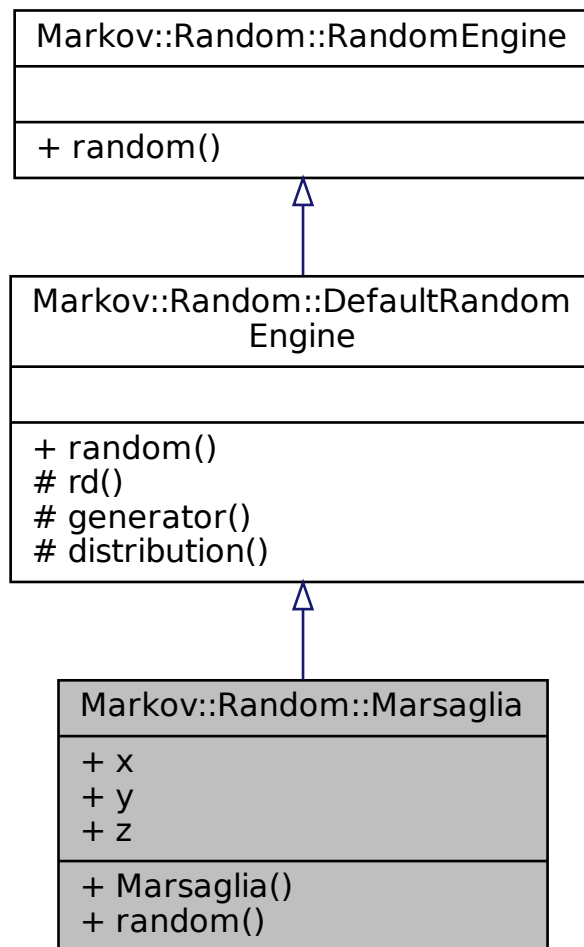
Implementation of [Marsaglia Random](#) Engine.

```
#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.

7.9.1 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](#).

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Marsaglia()

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

Construct [Marsaglia](#) Engine.

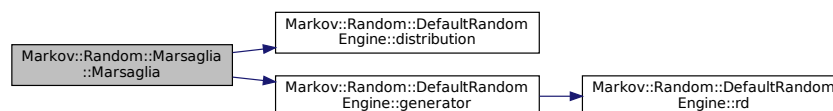
Initialize x,y and z using the default random engine.

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:



7.9.3 Member Function Documentation

7.9.3.1 distribution()

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

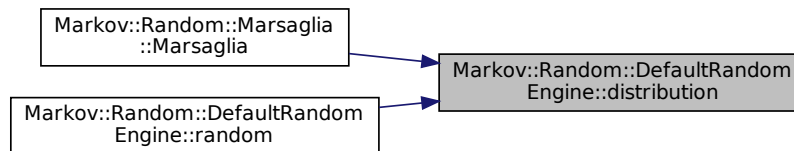
Distribution schema for seeding.

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:



7.9.3.2 generator()

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

Default random engine for seeding.

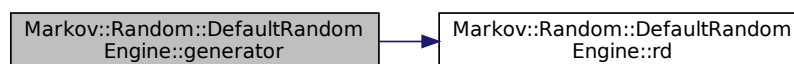
Definition at line 73 of file [random.h](#).

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

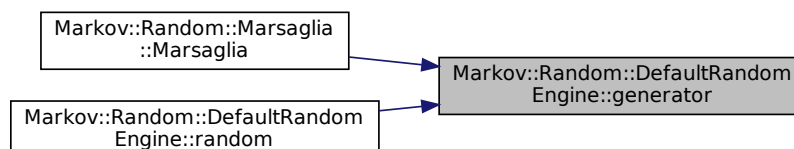
References [Markov::Random::DefaultRandomEngine::rd\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.9.3.3 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 129 of file [random.h](#).

```

00129                                     {
00130         unsigned long t;
00131         x ^= x << 16;
00132         x ^= x >> 5;
00133         x ^= x << 1;
00134
00135         t = x;
00136         x = y;
00137         y = z;
00138         z = t ^ x ^ y;
00139
00140         return z;
00141     }

```

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

7.9.3.4 rd()

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

Default random device for seeding.

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

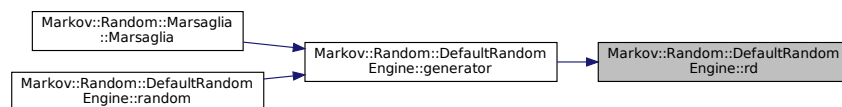
```

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:



7.9.4 Member Data Documentation

7.9.4.1 x

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

Definition at line 144 of file [random.h](#).

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

7.9.4.2 y

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

Definition at line 145 of file [random.h](#).

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

7.9.4.3 z

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

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

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

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

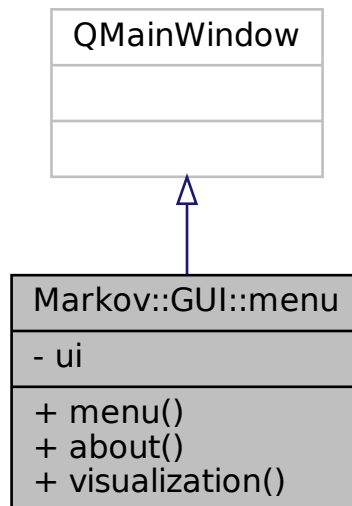
- [random.h](#)

7.10 Markov::GUI::menu Class Reference

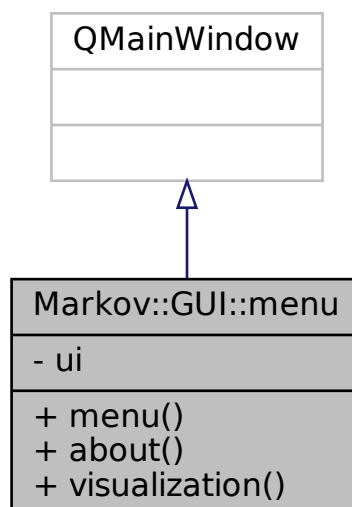
QT Menu class.

```
#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](#)

7.10.1 Detailed Description

QT Menu class.

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

7.10.2 Constructor & Destructor Documentation

7.10.2.1 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 }
```

References [ui](#).

7.10.3 Member Function Documentation

7.10.3.1 about

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

Definition at line 17 of file [menu.cpp](#).

```
00017 {
00018
00019
00020 }
```

7.10.3.2 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 }
```

7.10.4 Member Data Documentation

7.10.4.1 ui

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

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

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

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

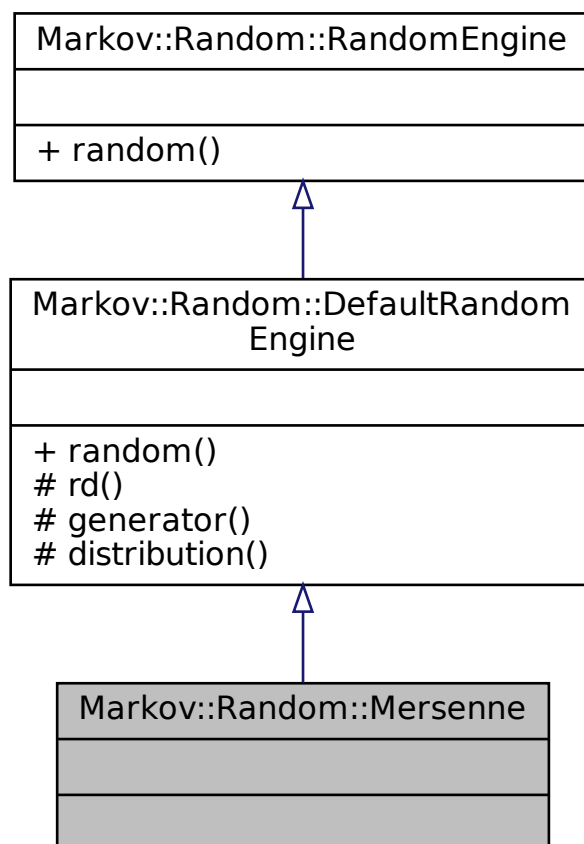
- [menu.h](#)
- [menu.cpp](#)

7.11 Markov::Random::Mersenne Class Reference

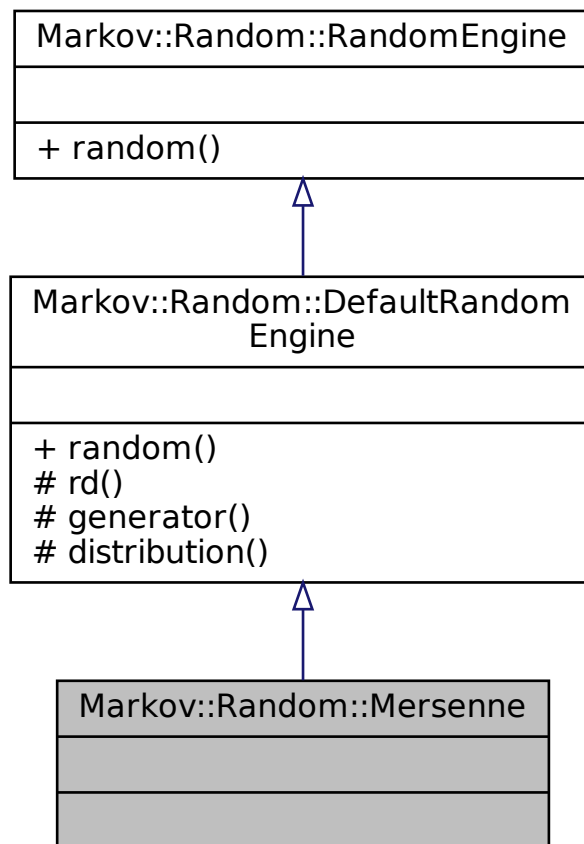
Implementation of [Mersenne](#) Twister Engine.

`#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.

7.11.1 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 174 of file [random.h](#).

7.11.2 Member Function Documentation

7.11.2.1 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](#).

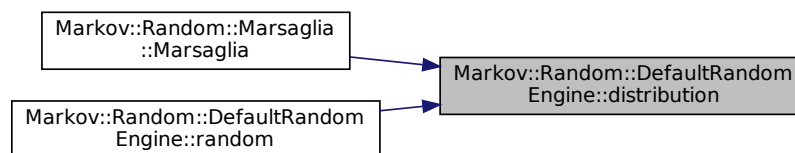
```

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:



7.11.2.2 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     }

```

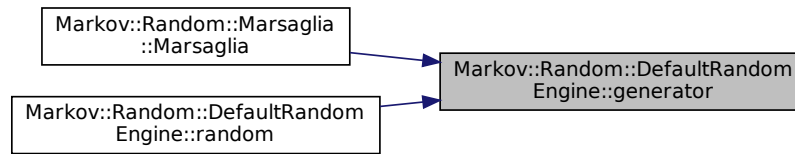
References [Markov::Random::DefaultRandomEngine::rd\(\)](#).

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:



7.11.2.3 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](#).

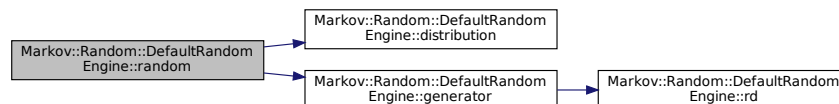
Definition at line 57 of file [random.h](#).

```

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:



7.11.2.4 rd()

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

Default random device for seeding.

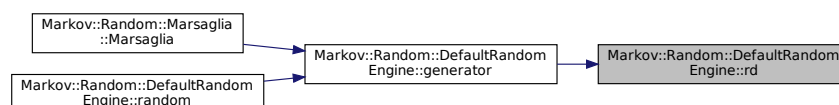
Definition at line 65 of file [random.h](#).

```

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:

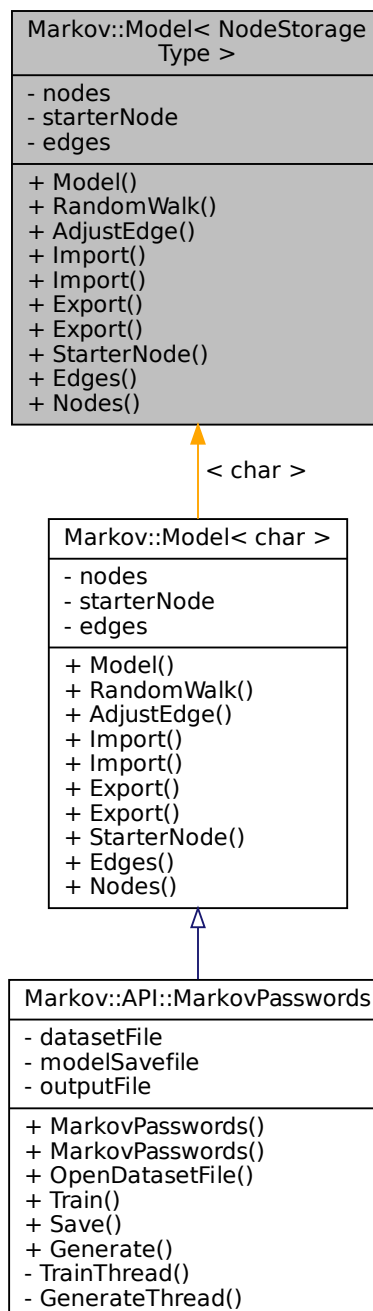
- [random.h](#)

7.12 Markov::Model< NodeStorageType > Class Template Reference

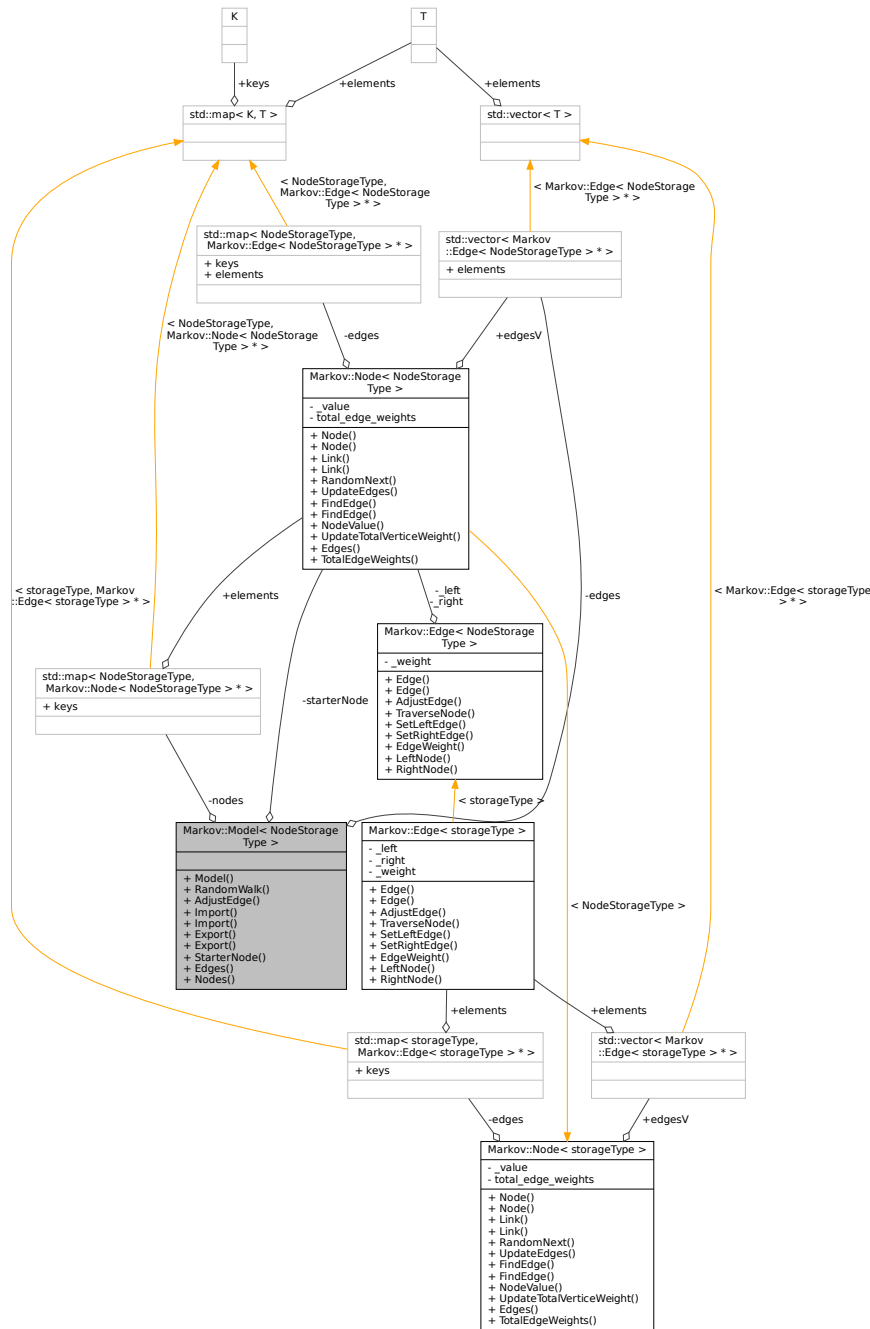
class for the final [Markov Model](#), constructed from nodes and edges.

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

Inheritance diagram for Markov::Model< NodeStorageType >:



Collaboration diagram for Markov::Model< NodeStorageType >:



Public Member Functions

- [Model](#) ()
Initialize a model with only start and end nodes.
- NodeStorageType * [RandomWalk](#) (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, NodeStorageType *buffer)
Do a random walk on this model.
- void [AdjustEdge](#) (const NodeStorageType *payload, long int occurrence)
Adjust the model with a single string.
- bool [Import](#) (std::ifstream *)

- *Import a file to construct the model.*
• bool [Import](#) (const char *filename)
Open a file to import with filename, and call bool [Model::Import](#) with std::ifstream.
- bool [Export](#) (std::ofstream *)
Export a file of the model.
• bool [Export](#) (const char *filename)
Open a file to export with filename, and call bool [Model::Export](#) with std::ofstream.
- [Node](#)< NodeStorageType > * [StarterNode](#) ()
Return starter [Node](#).
- std::vector< [Edge](#)< NodeStorageType > * > * [Edges](#) ()
Return a vector of all the edges in the model.
- std::map< NodeStorageType, [Node](#)< NodeStorageType > * > * [Nodes](#) ()
Return starter [Node](#).

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.

7.12.1 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](#).

7.12.2 Constructor & Destructor Documentation

7.12.2.1 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 }
```

7.12.3 Member Function Documentation

7.12.3.1 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 }
```

7.12.3.2 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; }
```

7.12.3.3 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 }
```

7.12.3.4 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

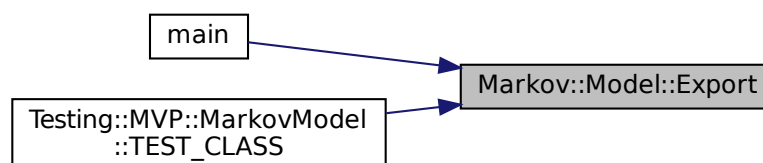
```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

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";
00279     }
00280
00281     return true;
00282 }
```

Referenced by [main\(\)](#), and [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#).

Here is the caller graph for this function:

**7.12.3.5 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 }
```

7.12.3.6 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 }
```

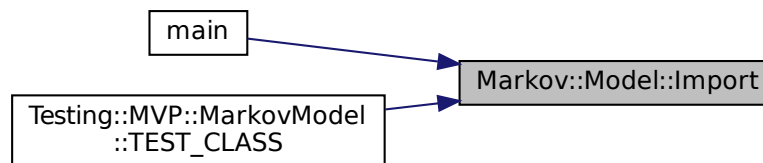
```

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 }

```

Referenced by [main\(\)](#), and [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#).

Here is the caller graph for this function:



7.12.3.7 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; }
```

7.12.3.8 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 criteria is reached, ignore the last selection and re-invoke randomNext

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

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

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

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

Parameters

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

Returns

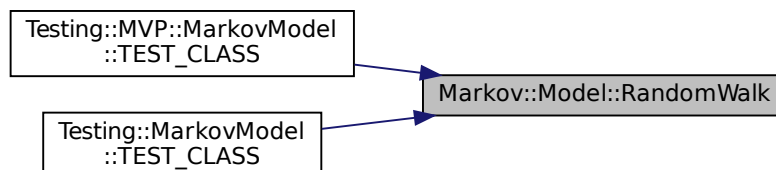
Null terminated string that was generated.

Definition at line 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
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 }
```

Referenced by [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#), and [Testing::MarkovModel::TEST_CLASS\(\)](#).

Here is the caller graph for this function:



7.12.3.9 StarterNode()

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

Returns

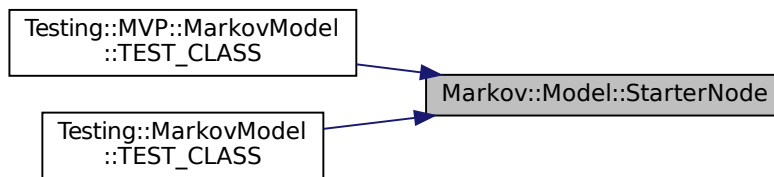
starter node with 00 NodeValue

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

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

Referenced by [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#), and [Testing::MarkovModel::TEST_CLASS\(\)](#).

Here is the caller graph for this function:



7.12.4 Member Data Documentation

7.12.4.1 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\(\)](#).

7.12.4.2 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\(\)](#).

7.12.4.3 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:

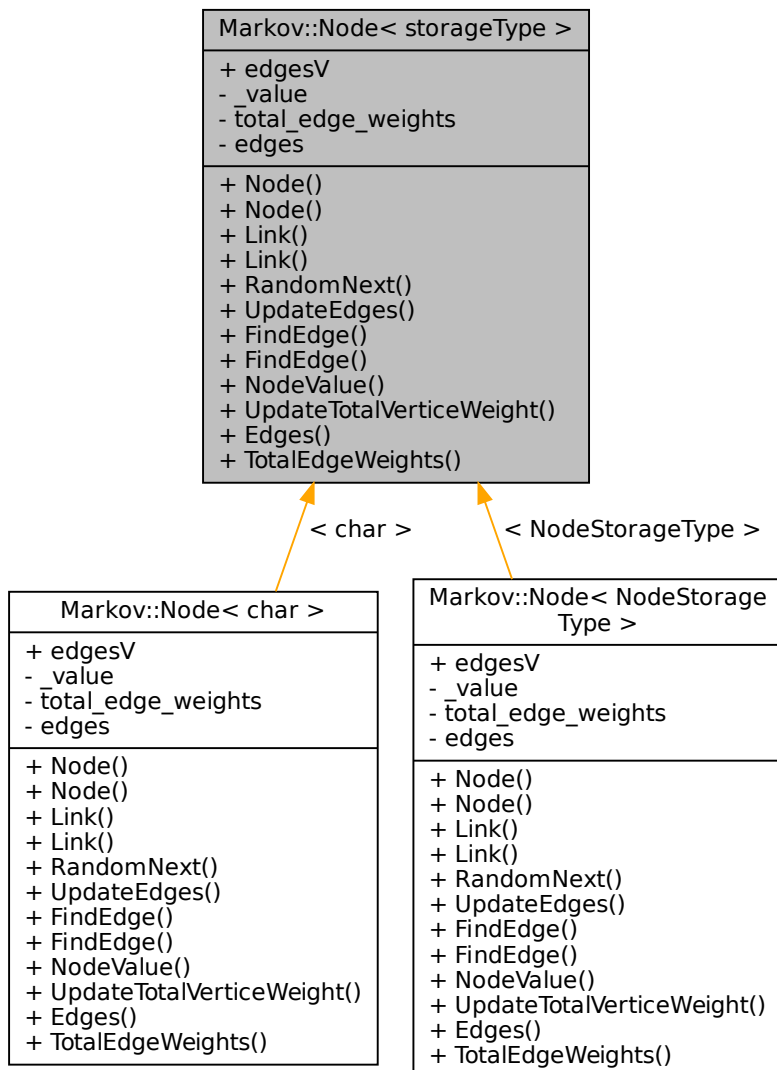
- [model.h](#)

7.13 Markov::Node< storageType > Class Template Reference

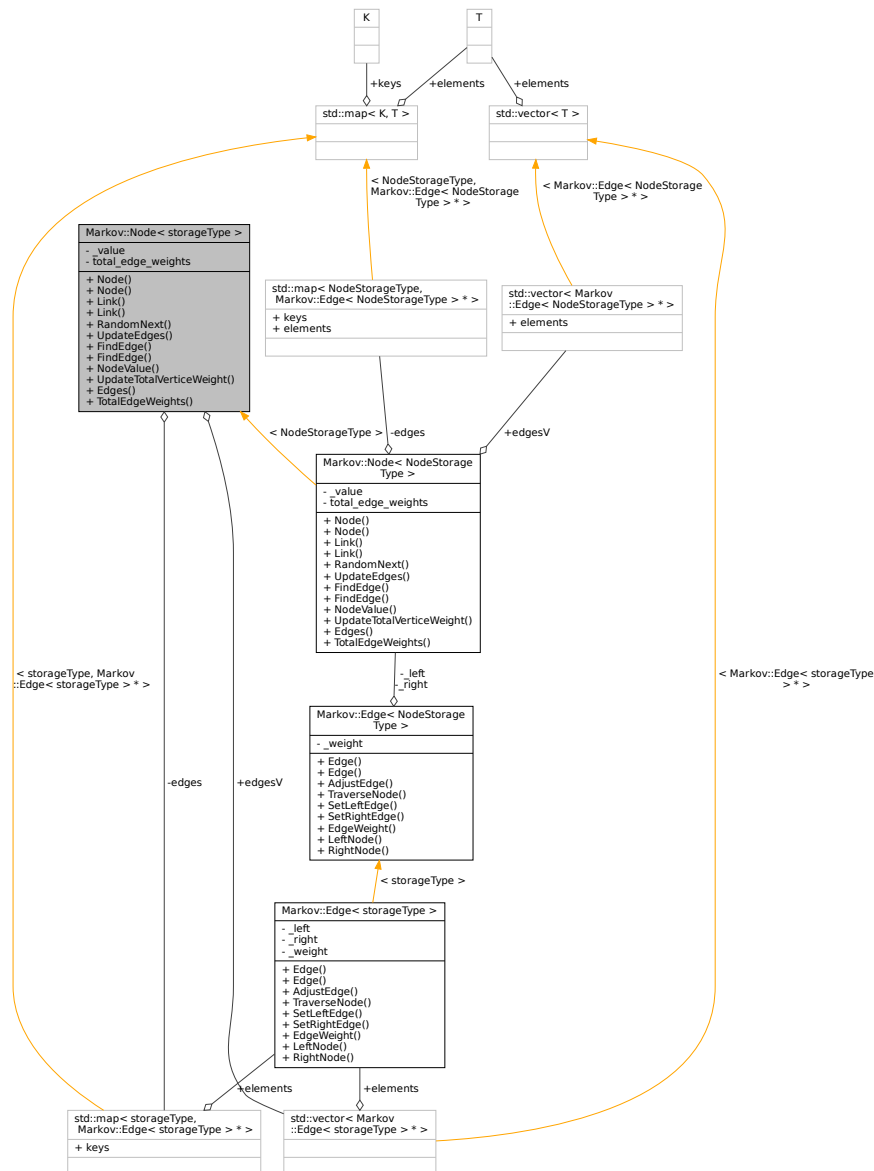
A node class that for the vertices of model. Connected with eachother using [Edge](#).

```
#include <edge.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;.

7.13.1 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 8 of file [edge.h](#).

7.13.2 Constructor & Destructor Documentation

7.13.2.1 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 };
```

7.13.2.2 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 };
```

7.13.3 Member Function Documentation

7.13.3.1 Edges()

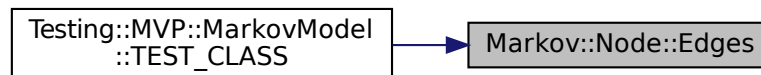
```
template<typename storageType >
std::map< storageType, Markov::Edge< storageType > * > * Markov::Node< storageType >::Edges
[inline]
return edges
```

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

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

Referenced by [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#).

Here is the caller graph for this function:



7.13.3.2 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

<code>target</code>	- target node.
---------------------	----------------

Returns

[Edge](#) that is connected between this node, and the target node.

7.13.3.3 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

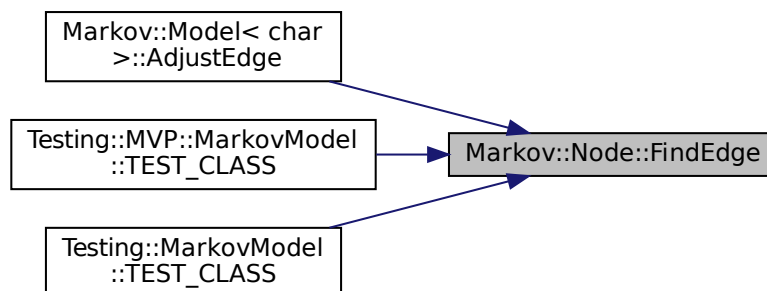
```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
Markov::Edge<unsigned char>* res = NULL;
src->Link(target1);
src->Link(target2);
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 };
```

Referenced by [Markov::Model< char >::AdjustEdge\(\)](#), [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#), and [Testing::MarkovModel::TEST_CLASS\(\)](#).

Here is the caller graph for this function:



7.13.3.4 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

<i>Edge</i>	- 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 }
```

7.13.3.5 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

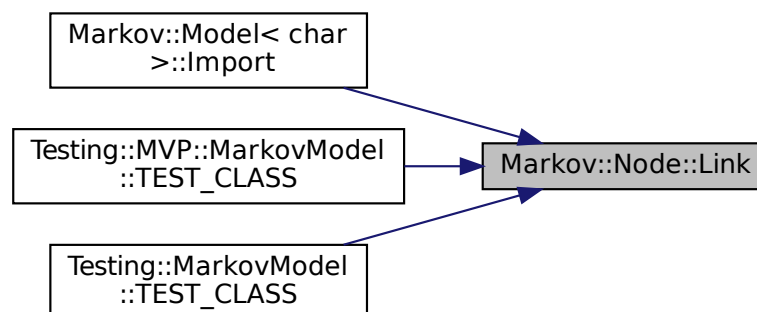
```
Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
```

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 }
```

Referenced by [Markov::Model< char >::Import\(\)](#), [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#), and [Testing::MarkovModel::TEST_C](#)

Here is the caller graph for this function:



7.13.3.6 NodeValue()

```
template<typename storageType >
unsigned char Markov::Node< storageType >::NodeValue [inline]
Return character representation of this node.
```

Returns

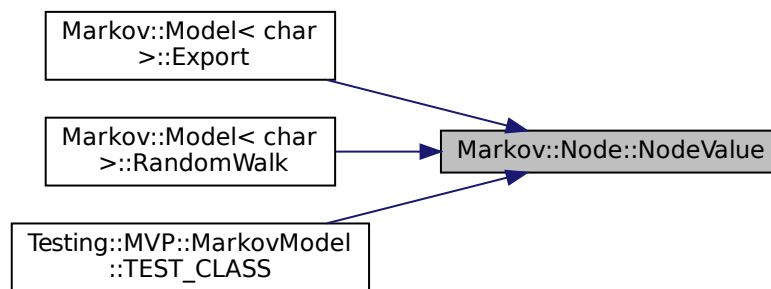
character representation at `_value`.

Definition at line 202 of file `node.h`.

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

Referenced by `Markov::Model< char >::Export()`, `Markov::Model< char >::RandomWalk()`, and `Testing::MVP::MarkovModel::TEST_`

Here is the caller graph for this function:



7.13.3.7 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](#).

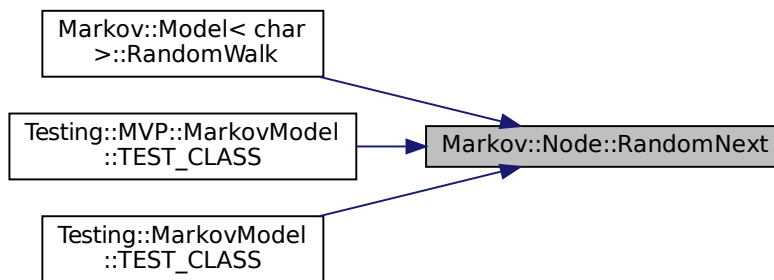
```

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 }

```

Referenced by [Markov::Model< char >::RandomWalk\(\)](#), [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#), and [Testing::MarkovModel::TEST_CLASS\(\)](#).

Here is the caller graph for this function:



7.13.3.8 TotalEdgeWeights()

```

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

```

Definition at line 264 of file [node.h](#).

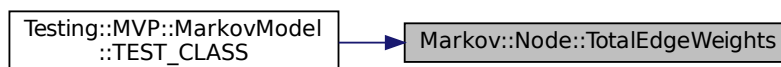
```

00264
00265     return this->total_edge_weights;
00266 }

```

Referenced by [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#).

Here is the caller graph for this function:



7.13.3.9 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 }
```

Referenced by [Testing::MVP::MarkovModel::TEST_CLASS\(\)](#).

Here is the caller graph for this function:



7.13.3.10 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 }
```

7.13.4 Member Data Documentation

7.13.4.1 `_value`

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

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

7.13.4.2 `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](#).

7.13.4.3 `edgesV`

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

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

7.13.4.4 `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:

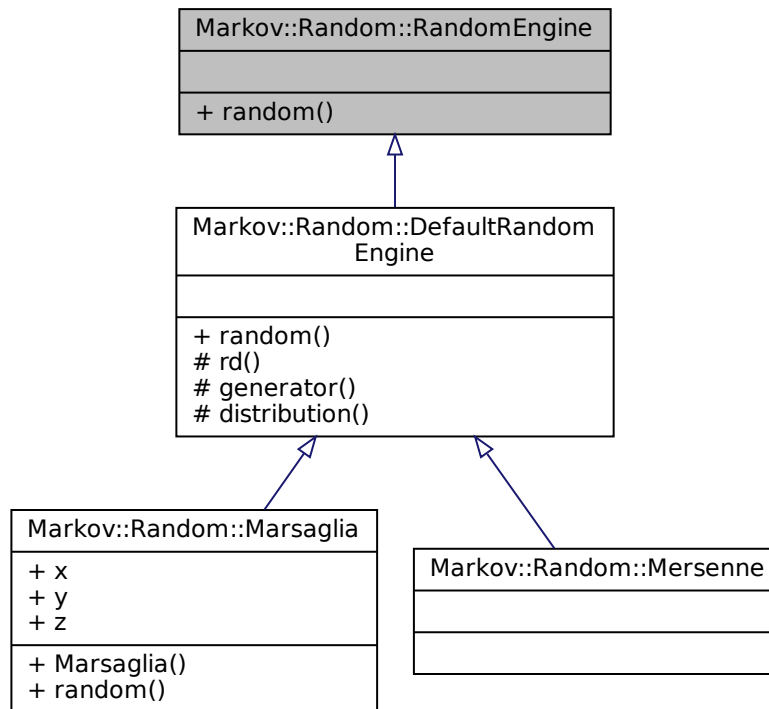
- [edge.h](#)
- [node.h](#)

7.14 Markov::Random::RandomEngine Class Reference

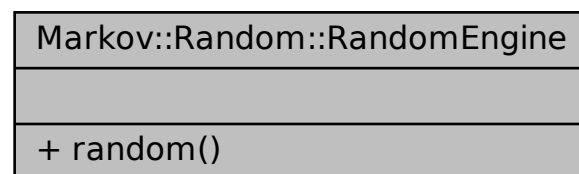
An abstract class for [Random](#) Engine.

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

Inheritance diagram for Markov::Random::RandomEngine:



Collaboration diagram for Markov::Random::RandomEngine:



Public Member Functions

- virtual unsigned long `random` ()=0

7.14.1 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`.

7.14.2 Member Function Documentation

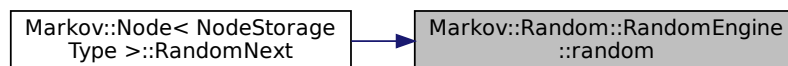
7.14.2.1 random()

```
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:

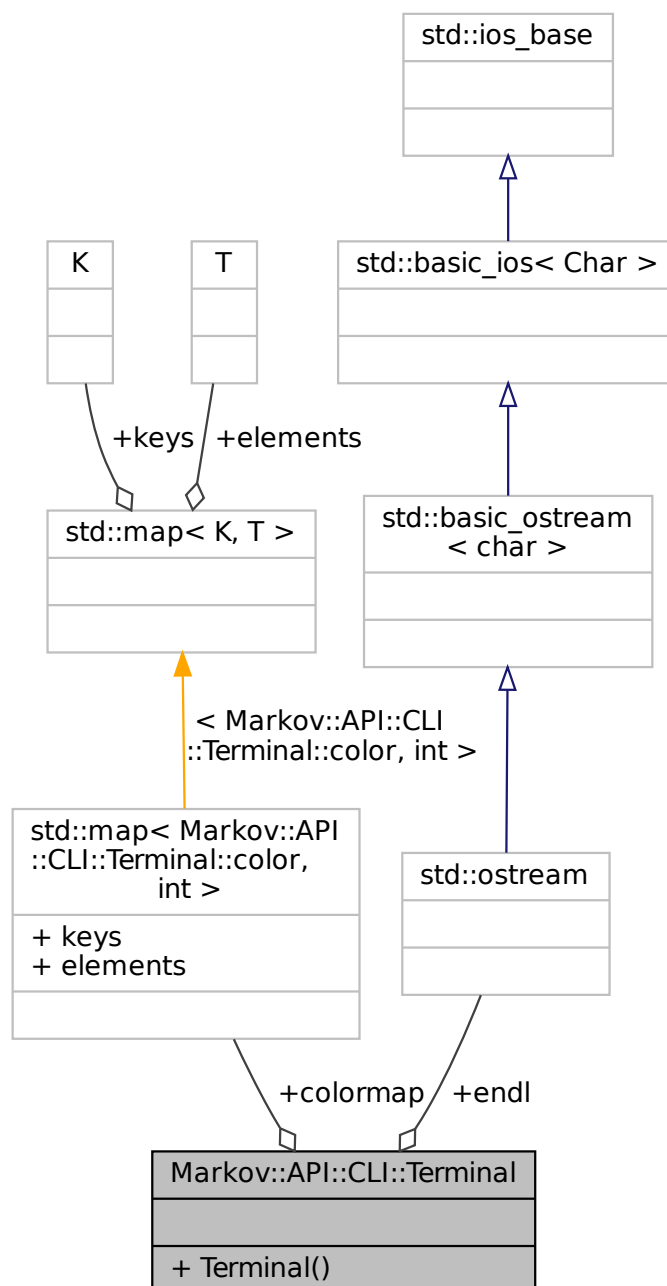
- [random.h](#)

7.15 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](#)

7.15.1 Detailed Description

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

7.15.2 Member Enumeration Documentation

7.15.2.1 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 };
```

7.15.3 Constructor & Destructor Documentation

7.15.3.1 Terminal()

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

Default constructor. Get references to stdout and stderr handles.

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

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

7.15.4 Member Data Documentation

7.15.4.1 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 [Markov::API::CLI::operator<<\(\)](#).

7.15.4.2 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:

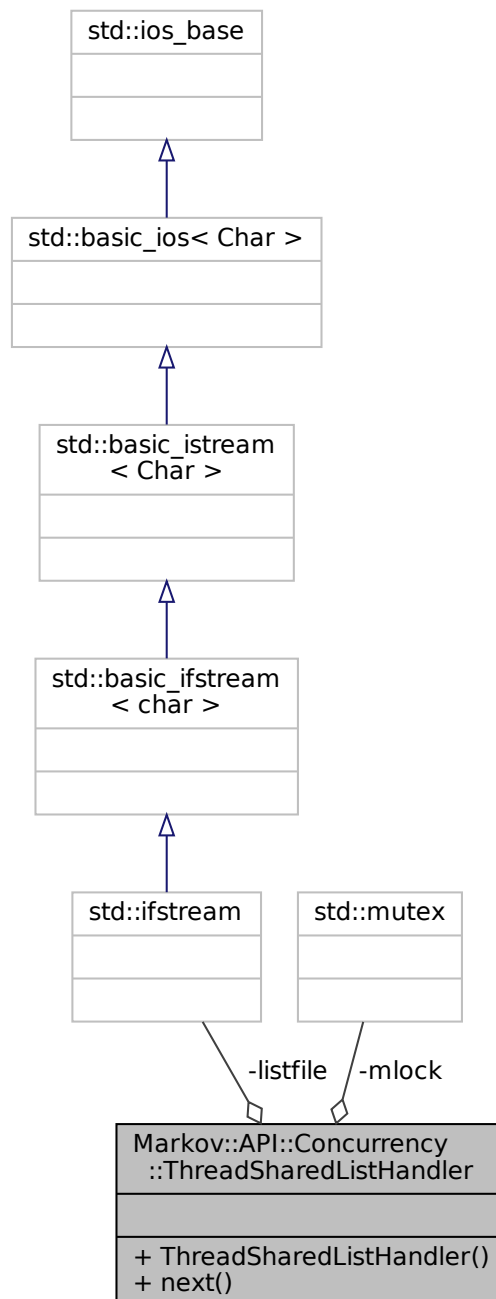
- [term.h](#)
- [term.cpp](#)

7.16 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](#)

7.16.1 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](#).

7.16.2 Constructor & Destructor Documentation

7.16.2.1 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](#).

7.16.3 Member Function Documentation

7.16.3.1 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

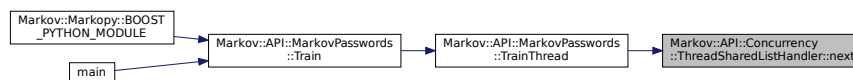
```
ThreadSharedListHandler listhandler("test.txt");
std::string line;
std::cout << listhandler->next(&line) << "\n";
```

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 }
```

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

Here is the caller graph for this function:



7.16.4 Member Data Documentation

7.16.4.1 listfile

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

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

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

7.16.4.2 mlock

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

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

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

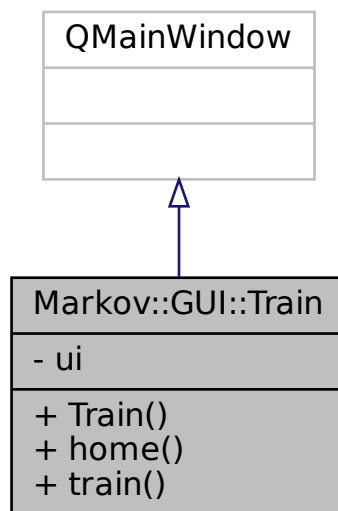
- [threadSharedListHandler.h](#)
- [threadSharedListHandler.cpp](#)

7.17 Markov::GUI::Train Class Reference

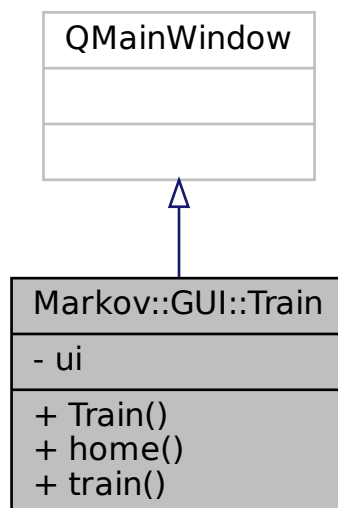
QT Training page class.

```
#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](#)

7.17.1 Detailed Description

QT Training page class.

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

7.17.2 Constructor & Destructor Documentation

7.17.2.1 Train()

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

7.17.3 Member Function Documentation

7.17.3.1 home

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

7.17.3.2 train

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

7.17.4 Member Data Documentation

7.17.4.1 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](#)

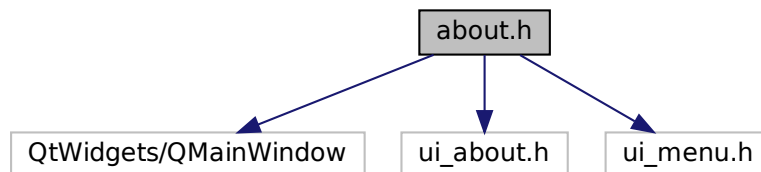
Chapter 8

File Documentation

8.1 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

8.2 about.h

```
00001 #pragma once
00002 #include <QtWidgets/QMainWindow>
00003 #include "ui_about.h"
00004 #include <ui_menu.h>
00005
00008 namespace Markov::GUI{
00009
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 };
00023
00024

```

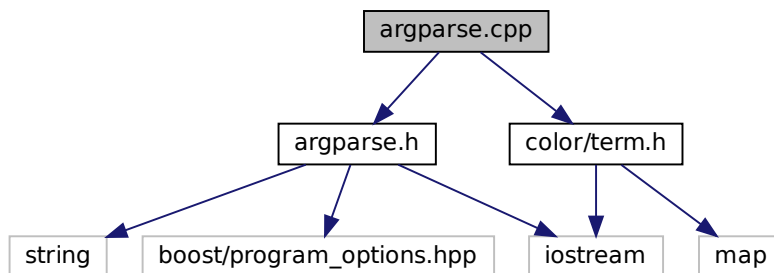
8.3 argparse.cpp File Reference

```

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

```

Include dependency graph for argparse.cpp:



8.4 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
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 }
00031

```

8.5 argparse.h File Reference

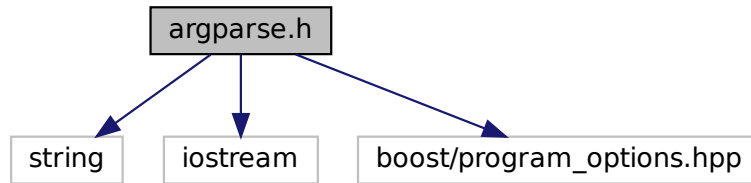
```

#include <string>

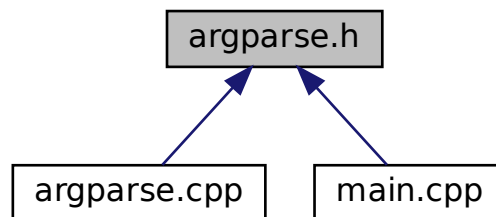
```

```
#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.

8.5.1 Macro Definition Documentation

8.5.1.1 BOOST_ALL_DYN_LINK

```
#define BOOST_ALL_DYN_LINK 1
```

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

8.6 argparse.h

```
00001 #include<string>
00002 #include<iostream>
00003
00004 #define BOOST_ALL_DYN_LINK 1
00005
00006 #include <boost/program_options.hpp>
00009 namespace opt = boost::program_options;
00010
00014 namespace Markov::API::CLI{
00015
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
00034     class Argparse {
00035     public:
00036
00037         Argparse();
00038
00046         Argparse(int argc, char** argv) {
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
00067 when done. Will be ignored for generation mode")
00068                 ("dataset", opt::value<std::string>(), "Dataset file to read input from training. Will
be ignored for generation mode")
00069                 ("seperator", opt::value<char>(), "Seperator character to use with training data.
(character between occurence and value)")
00070                 ("wordlist", opt::value<std::string>(), "Wordlist file path to export generation
results to. Will be ignored for training mode")
00071                 ("count", opt::value<int>(), "Number of lines to generate. Ignored in training mode")
00072                 ("verbosity", "Output verbosity")
00073                 ("help", "Option definitions");
00074
00075             opt::variables_map vm;
```

```

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
00155     Markov::API::CLI::ProgramOptions getProgramOptions(void) {
00156         return this->po;
00157     }
00158
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
00195     static Markov::API::CLI::ProgramOptions* parse(int argc, char** argv);
00196
00197
00200     static void help();
00201
00202     private:
00203     Markov::API::CLI::ProgramOptions po;
00204 };
00205
00206 };
00207

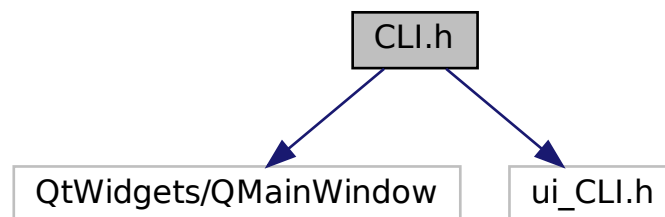
```

8.7 CLI.h File Reference

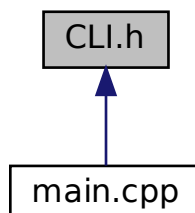
```
#include <QtWidgets/QMainWindow>
```

```
#include "ui_CLI.h"
```

Include dependency graph for CLI.h:



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



Classes

- class [Markov::GUI::CLI](#)
QT [CLI](#) Class.

Namespaces

- [Markov](#)

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

- [Markov::GUI](#)

namespace for MarkovPasswords [API GUI](#) wrapper

8.8 CLI.h

```

00001 #pragma once
00002 #include <QtWidgets/QMainWindow>
00003 #include "ui_CLI.h"
00004
00005 namespace Markov::GUI{
00006     class CLI :public QMainWindow {
00007     Q_OBJECT
00008     public:
00009         CLI(QWidget* parent = Q_NULLPTR);
00010
00011     private:
00012         Ui::CLI ui;
00013
00014     public slots:
00015         void start();
00016         void statistics();
00017         void about();
00018     };
00019 };
00020
00021

```

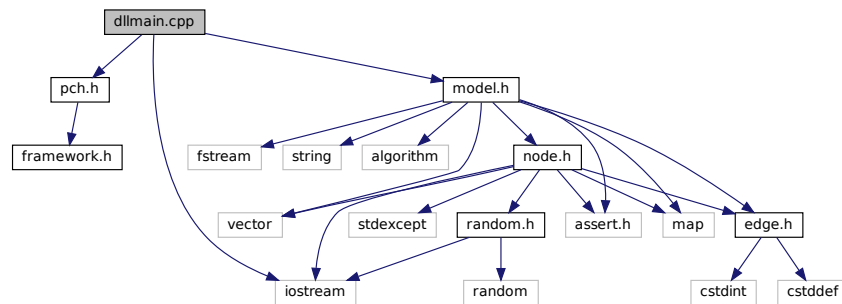
8.9 dllmain.cpp File Reference

```

#include "pch.h"
#include "model.h"
#include <iostream>

```

Include dependency graph for dllmain.cpp:



8.10 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
00026

```

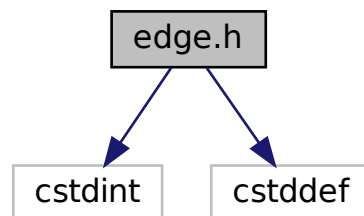
8.11 edge.h File Reference

```

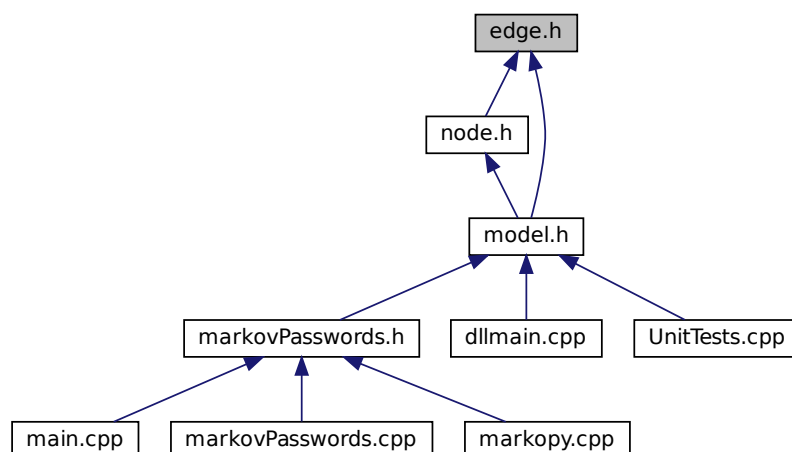
#include <stdint>
#include <stddef>

```

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.

8.12 edge.h

```

00001 #pragma once
00002 #include <cstdint>
00003 #include <cstddef>
00004
00005 namespace Markov {
00006
00007     template <typename NodeStorageType>
00008     class Node;
00009
00010     template <typename NodeStorageType>
00011     class Edge {
00012     public:
00013
00014         Edge<NodeStorageType>();
00015
00016         Edge<NodeStorageType>(Node<NodeStorageType>* _left, Node<NodeStorageType>* _right);
00017
00018         void AdjustEdge(long int offset);
00019
00020         inline Node<NodeStorageType>* TraverseNode();
00021
00022         void SetLeftEdge (Node<NodeStorageType>*);
00023         void SetRightEdge (Node<NodeStorageType>*);
00024
00025         inline uint64_t EdgeWeight();
00026
00027         Node<NodeStorageType>* LeftNode();
00028
00029         inline Node<NodeStorageType>* RightNode();
00030
00031     private:
00032         Node<NodeStorageType>* _left;
00033         Node<NodeStorageType>* _right;
00034         long int _weight;
00035     };
00036 };
00037
00038 //default constructor of edge
00039 template <typename NodeStorageType>
00040 Markov::Edge<NodeStorageType>::Edge() {
00041     this->_left = NULL;
00042     this->_right = NULL;
00043     this->_weight = 0;
00044 }
00045
00046 //constructor of edge
00047 template <typename NodeStorageType>
00048 Markov::Edge<NodeStorageType>::Edge(Markov::Node<NodeStorageType>* _left,
00049     Markov::Node<NodeStorageType>* _right) {
00050     this->_left = _left;
00051     this->_right = _right;
00052     this->_weight = 0;
00053 }
00054
00055 //to AdjustEdge the edges by the edge with its offset
00056 template <typename NodeStorageType>
00057 void Markov::Edge<NodeStorageType>::AdjustEdge(long int offset) {
00058     this->_weight += offset;
00059     this->LeftNode()->UpdateTotalVerticeWeight(offset);
00060 }
00061
00062 //to TraverseNode the node
00063 template <typename NodeStorageType>
00064 inline Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::TraverseNode() {
00065     if (this->RightNode()->NodeValue() == 0xff) //terminator node
00066         return NULL;
00067     return _right;
00068 }
00069
00070 //to set the LeftNode of the node
00071 template <typename NodeStorageType>
00072 void Markov::Edge<NodeStorageType>::SetLeftEdge(Markov::Node<NodeStorageType>* n) {
00073     this->_left = n;
00074 }
00075
00076 //to set the RightNode of the node

```

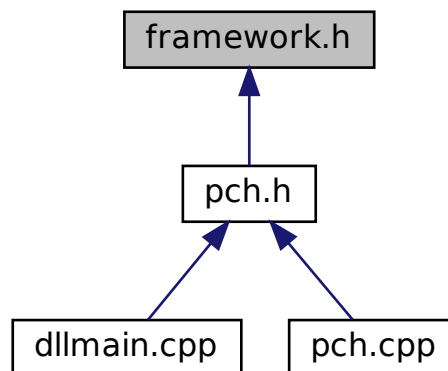
```

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 }
00155
00156

```

8.13 framework.h File Reference

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



Macros

- `#define WIN32_LEAN_AND_MEAN`

8.13.1 Macro Definition Documentation

8.13.1.1 WIN32_LEAN_AND_MEAN

```
#define WIN32_LEAN_AND_MEAN
```

Definition at line 3 of file [framework.h](#).

8.14 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
00009

```

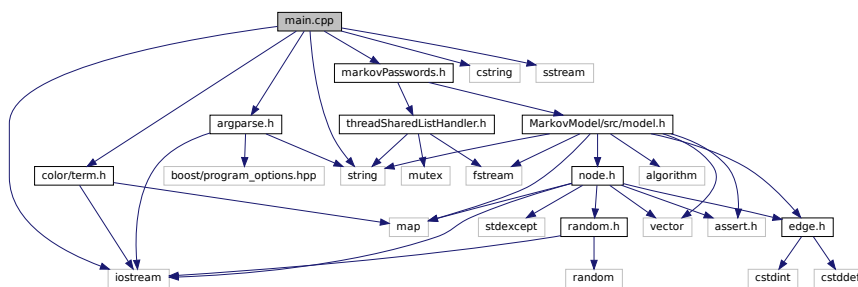
8.15 main.cpp File Reference

```

#include <iostream>
#include "color/term.h"
#include "argparse.h"
#include <string>
#include <cstring>
#include <sstream>
#include "markovPasswords.h"

```

Include dependency graph for src/main.cpp:



Functions

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

8.15.1 Function Documentation

8.15.1.1 main()

```

int main (
    int argc,
    char ** argv )

```

Launch CLI tool.

Definition at line 13 of file [src/main.cpp](#).

```

00013     {
00014
00015         Markov::API::CLI::Terminal t;
00016         /*
00017         ProgramOptions* p = Argparse::parse(argc, argv);
00018
00019         if (p==0 || p->bFailure) {
00020             std::cout << TERM_FAIL << "Arguments Failed to Parse" << std::endl;
00021             Argparse::help();
00022         }*/
00023         Markov::API::CLI::Argparse a(argc,argv);
00024
00025         Markov::API::MarkovPasswords markovPass;
00026         std::cout << "Importing model.\n";
00027         markovPass.Import("models/2gram.mdl");
00028         std::cout << "Import done. Training...\n";
00029         markovPass.Train("datasets/graduation.corpus", '\t', 50);
00030         std::cout << "Training done. Exporting to file.\n";
00031         markovPass.Export("models/finished2.mdl");

```

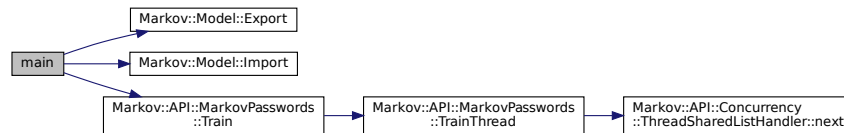
```

00032
00033     std::cout << "Generation done...\n";
00034     return 0;
00035 }

```

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

Here is the call graph for this function:



8.16 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
00010
00013 int main(int argc, char** argv) {
00014
00015     Markov::API::CLI::Terminal t;
00016     /*
00017     ProgramOptions* p = Argparse::parse(argc, argv);
00018
00019     if (p==0 || p->bFailure) {
00020         std::cout << TERM_FAIL << "Arguments Failed to Parse" << std::endl;
00021         Argparse::help();
00022     }*/
00023     Markov::API::CLI::Argparse a(argc,argv);
00024
00025     Markov::API::MarkovPasswords markovPass;
00026     std::cout << "Importing model.\n";
00027     markovPass.Import("models/2gram.mdl");
00028     std::cout << "Import done. Training...\n";
00029     markovPass.Train("datasets/graduation.corpus", '\t', 50);
00030     std::cout << "Training done. Exporting to file.\n";
00031     markovPass.Export("models/finished2.mdl");
00032
00033     std::cout << "Generation done...\n";
00034     return 0;
00035 }
00036

```

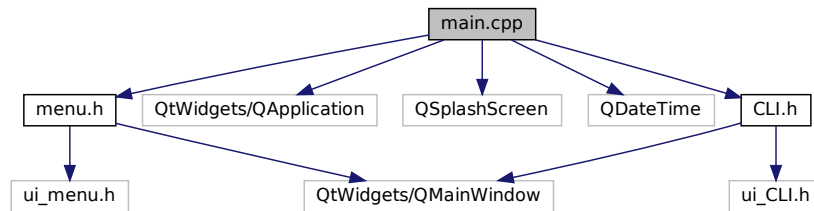
8.17 main.cpp File Reference

```

#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.

8.17.1 Function Documentation

8.17.1.1 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 }
```

8.18 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
00012 int main(int argc, char *argv[])
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 }
00036

```

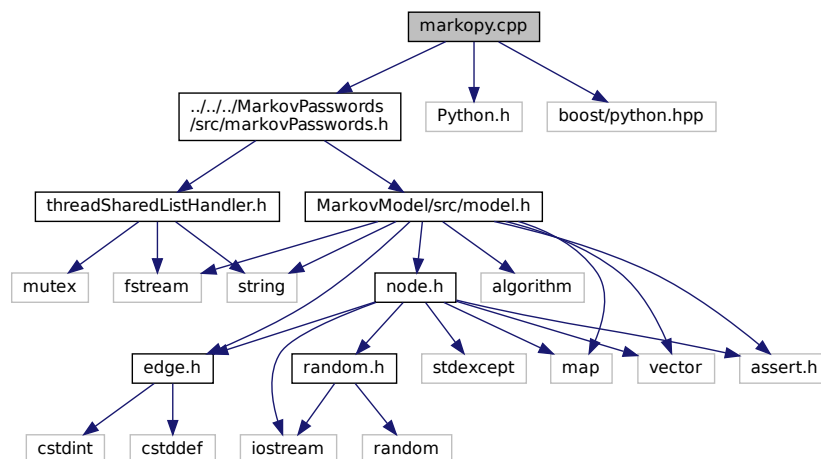
8.19 markopy.cpp File Reference

```

#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)

8.19.1 Macro Definition Documentation

8.19.1.1 BOOST_PYTHON_STATIC_LIB

#define BOOST_PYTHON_STATIC_LIB

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

8.20 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 };
00036
00037

```

8.21 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`

8.22 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: -vvvvvvvvvvvvvv")
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")

```



```

00056         + "Selected operation (generate/train) will be applied to each file in the folder,
and "
00057         + "output to the output directory.")
00058 args = parser.parse_args()
00059
00060
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
directory")
00075         exit(1)
00076
00077     model.Import(input_model)
00078     logging.pprint("Model imported successfully.", 2)
00079     return model
00080
00081
00082 def cli_train(model, dataset, separator, output, output_forced=False, bulk=False):
00083     if not (dataset and separator and (output or not output_forced)):
00084         logging.pprint(
00085             f"Training mode requires -d/--dataset{' ', -o/--output' if output_forced else ''} and
-s/--separator 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 separator.", 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.separator,
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",
2)
00151
00152                     model = cli_init(f"{args.input}/{input}")
00153                     model_base = input
00154                     if "." in args.input:
00155                         model_base = input.split(".")[1]
00156                     cli_generate(model, f"{args.wordlist}/{model_base}.txt", bulk=True)
00157         else:
00158             logging.pprint("In bulk generation, input and wordlist should be directory.")
00159
00160     else:
00161         model = cli_init(args.input)
00162         if (args.mode.lower() == "generate"):
00163             cli_generate(model, args.wordlist)
00164
00165         elif (args.mode.lower() == "train"):
00166             cli_train(model, args.dataset, args.seperator, args.output, output_forced=True)
00167
00168         elif (args.mode.lower() == "combine"):
00169             cli_train(model, args.dataset, args.seperator, args.output)
00170             cli_generate(model, args.wordlist)
00171
00172         else:
00173             logging.pprint("Invalid mode arguement given.")
00174             logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00175             exit(5)

```

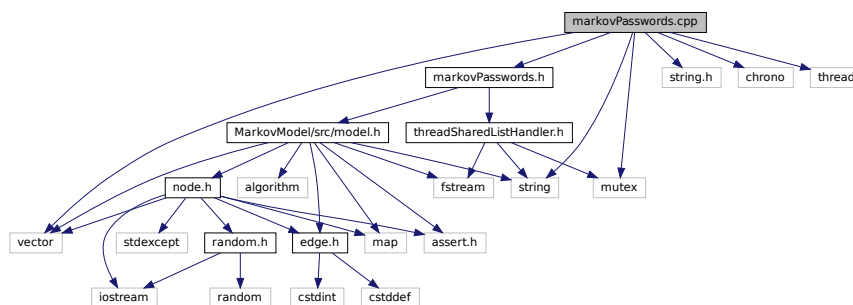
8.23 markovPasswords.cpp File Reference

```

#include "markovPasswords.h"
#include <string.h>
#include <chrono>
#include <thread>
#include <vector>
#include <mutex>
#include <string>

```

Include dependency graph for markovPasswords.cpp:



8.24 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     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
    *listhandler, char delimiter){
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 }
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
minLen, int maxLen, int threads) {
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 }
00113
00114 void Markov::API::MarkovPasswords::GenerateThread(std::mutex *outputLock, unsigned long int n,
std::ofstream *wordlist, int minLen, int maxLen) {
00115     char* res = new char[maxLen+5];
00116     if(n==0) return;
00117
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 }

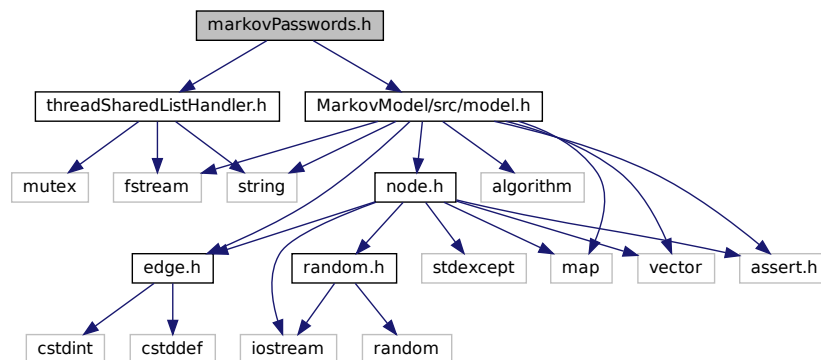
```

8.25 markovPasswords.h File Reference

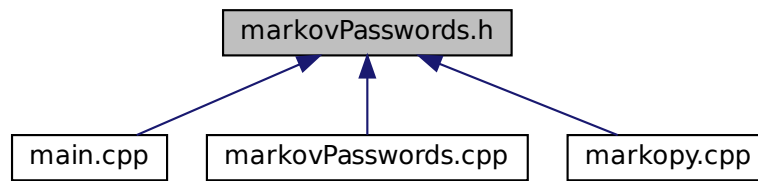
```
#include "threadSharedListHandler.h"
```

```
#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.

8.26 markovPasswords.h

```

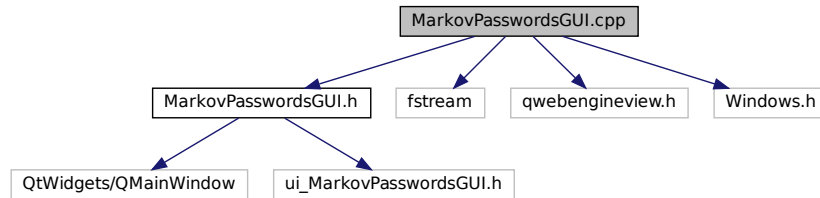
00001 #pragma once
00002 #include "threadSharedListHandler.h"
00003 #include "MarkovModel/src/model.h"
00004
00005
00008 namespace Markov::API{
00009
00017     class MarkovPasswords : public Markov::Model<char>{
00018     public:
00019
00024         MarkovPasswords();
00025
00039         MarkovPasswords(const char* filename);
00040
00045         std::ifstream* OpenDatasetFile(const char* filename);
00046
00047
00059         void Train(const char* datasetFileName, char delimiter, int threads);
00060
00061
00062
00067         std::ofstream* Save(const char* filename);
00068
00079         void Generate(unsigned long int n, const char* wordlistFileName, int minLen=6, int maxLen=12,
00080 int threads=20);
00081
00082     private:
00083
00089         void TrainThread(Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char
00090 delimiter);
00091
00099         void GenerateThread(std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int
00100 minLen, int maxLen);
00101         std::ifstream* datasetFile;
00102         std::ofstream* modelSavefile;
00103         std::ofstream* outputFile;
00104     };
00105
00106
00107 };
00108

```

8.27 MarkovPasswordsGUI.cpp File Reference

```
#include "MarkovPasswordsGUI.h"
#include <fstream>
#include <qwebengineview.h>
#include <Windows.h>
```

Include dependency graph for MarkovPasswordsGUI.cpp:



8.28 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     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {modelvisSelected(); });
00015     QObject::connect(ui.pushButton_4, &QPushButton::clicked, this, [this] {comparisonSelected(); });
00016 }
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 }

```

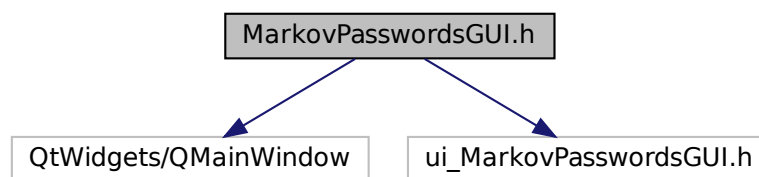
8.29 MarkovPasswordsGUI.h File Reference

```

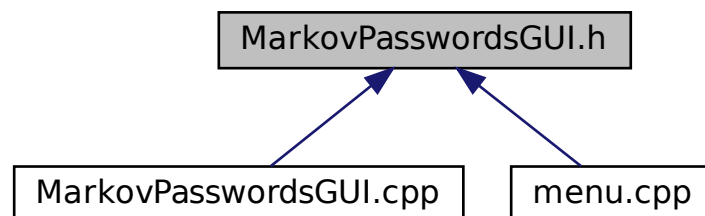
#include <QtWidgets/QMainWindow>
#include "ui_MarkovPasswordsGUI.h"

```

Include dependency graph for MarkovPasswordsGUI.h:



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



Classes

- class [Markov::GUI::MarkovPasswordsGUI](#)

Reporting UI.

Namespaces

- [Markov](#)

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

- [Markov::GUI](#)

namespace for MarkovPasswords [API GUI](#) wrapper

8.30 MarkovPasswordsGUI.h

```

00001 #pragma once
00002
00003 #include <QtWidgets/QMainWindow>
00004 #include "ui_MarkovPasswordsGUI.h"
00005
00006 namespace Markov::GUI{
00012     class MarkovPasswordsGUI : public QMainWindow {
00013         Q_OBJECT
00014
00015     public:
00019         MarkovPasswordsGUI(QWidget *parent = Q_NULLPTR);
00020
00024         void renderHTMLFile(std::string* filename);
00025
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 };
00044
00045

```

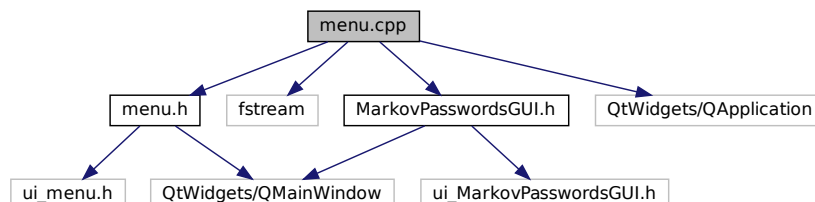
8.31 menu.cpp File Reference

```

#include "menu.h"
#include <fstream>
#include "MarkovPasswordsGUI.h"
#include <QtWidgets/QApplication>

```

Include dependency graph for menu.cpp:



8.32 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     //QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {about(); });
00014     QObject::connect(ui.visu, &QPushButton::clicked, this, [this] {visualization(); });
00015 }
00016
00017 void menu::about() {
00018
00019
00020 }
00021 void menu::visualization() {
00022     MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
00023     w->show();
00024     this->close();
00025 }

```

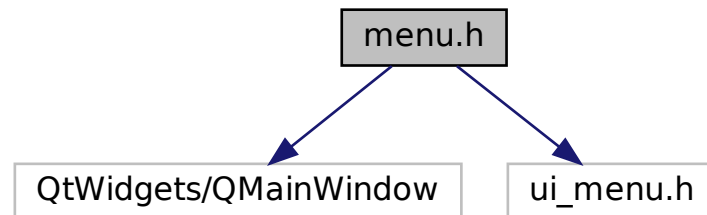
8.33 menu.h File Reference

```

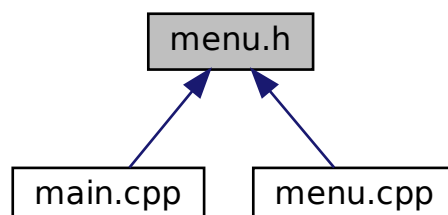
#include <QtWidgets/QMainWindow>
#include "ui_menu.h"

```

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

8.34 menu.h

```

00001 #pragma once
00002 #include <QtWidgets/QMainWindow>
00003 #include "ui_menu.h"
00004
00005
00006 namespace Markov::GUI{
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 };

```

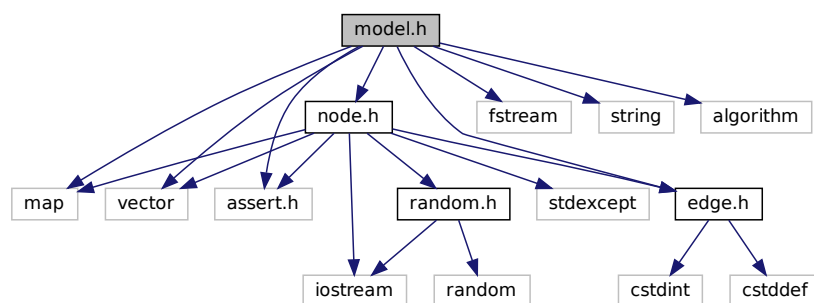
8.35 model.h File Reference

```

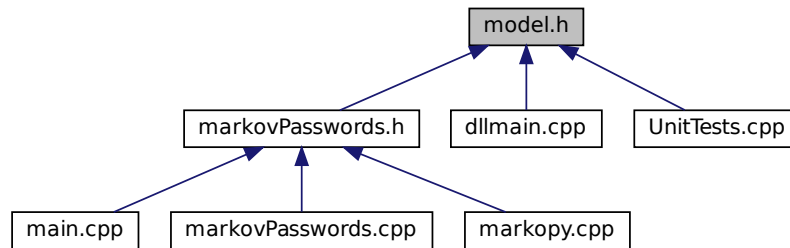
#include <map>
#include <vector>
#include <fstream>
#include <assert.h>
#include <string>
#include <algorithm>
#include "node.h"
#include "edge.h"

```

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.

8.36 model.h

```

00001
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
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
00041     class Model {
00042     public:
00043
00049         Model<NodeStorageType>();
00050
00082         NodeStorageType* RandomWalk(Markov::Random::RandomEngine* randomEngine, int minSetting, int
maxSetting, NodeStorageType* buffer);
00083
00105         void AdjustEdge(const NodeStorageType* payload, long int occurrence);
00106
00122         bool Import(std::ifstream*);
00123
00133         bool Import(const char* filename);
00134
00151         bool Export(std::ofstream*);
00152

```

```

00162         bool Export(const char* filename);
00163
00167         Node<NodeStorageType>* StarterNode(){ return starterNode;}
00168
00172         std::vector<Edge<NodeStorageType>*>* Edges(){ return &edges;}
00173
00177         std::map<NodeStorageType, Node<NodeStorageType>*>* Nodes(){ return &nodes;}
00178
00179     private:
00183         std::map<NodeStorageType, Node<NodeStorageType>*> nodes;
00184
00188         Node<NodeStorageType>* starterNode;
00189
00190
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
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 }
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
00272 template <typename NodeStorageType>
00273 bool Markov::Model<NodeStorageType>::Export(std::ofstream* f) {
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() <<
00280         "\n";
00281     }
00282     return true;
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*
00293     randomEngine, int minSetting, int maxSetting, NodeStorageType* buffer) {
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 template <typename NodeStorageType>
00318 void Markov::Model<NodeStorageType>::AdjustEdge(const NodeStorageType* payload, long int occurrence) {
00319     NodeStorageType p = payload[0];
00320     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00321     Markov::Edge<NodeStorageType>* e;
00322     int i = 0;
00323     if (p == 0) return;
00324     while (p != 0) {
00325         e = curnode->FindEdge(p);
00326         if (e == NULL) return;
00327         e->AdjustEdge(occurrence);
00328         curnode = e->RightNode();
00329         p = payload[++i];
00330     }
00331     e = curnode->FindEdge('\xff');
00332     e->AdjustEdge(occurrence);
00333     return;
00334 }
00335 }
00336
00337
00338
00339
00340
00341
00342
00343
00344

```

8.37 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

8.38 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")

```

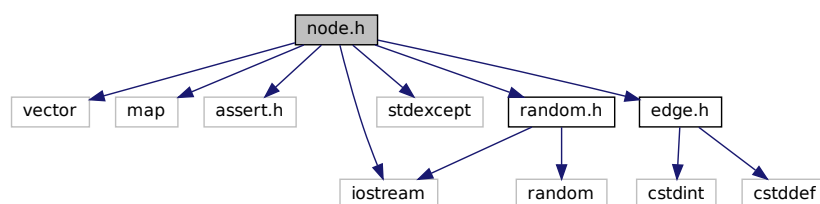
8.39 node.h File Reference

```

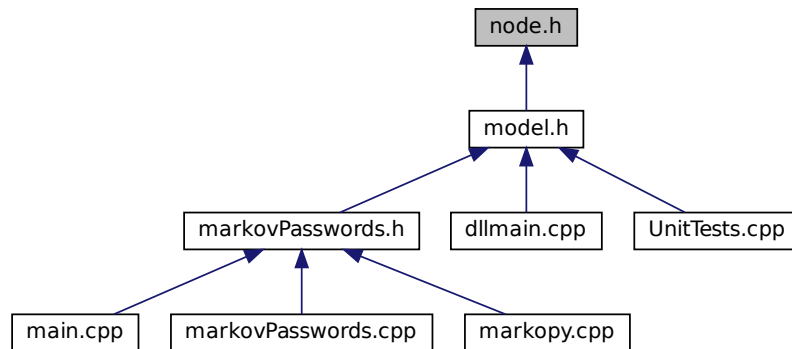
#include <vector>
#include <map>
#include <assert.h>
#include <iostream>
#include <stdexcept>
#include "edge.h"
#include "random.h"

```

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.

8.40 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
00015     template <typename storageType>
00016     class Node {
00017     public:
00018
00021         Node<storageType>();
00022
00032         Node<storageType>(storageType _value);
00033
00047         Edge<storageType>* Link(Node<storageType>*);
00048
00063         Edge<storageType>* Link(Edge<storageType>*);
00064
00098         Node<storageType>* RandomNext(Markov::Random::RandomEngine* randomEngine);
00099
00118         bool UpdateEdges(Edge<storageType>*);
00119
00138         Edge<storageType>* FindEdge(storageType repr);
00139
00144         Edge<storageType>* FindEdge(Node<storageType>* target);
00145
00149         inline unsigned char NodeValue();
00150
00154         void UpdateTotalVerticeWeight(long int offset);
00155
00158         inline std::map<storageType, Edge<storageType>*>* Edges();
00159
00162         inline long int TotalEdgeWeights();
00163
00164

```

```

00165         std::vector<Edge<storageType>*> edgesV;
00166     private:
00167
00168
00169         storageType _value;
00171         long int total_edge_weights;
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) {

```



```

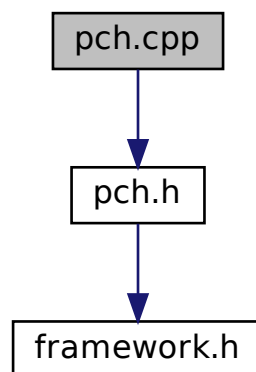
00255     this->total_edge_weights += offset;
00256 }
00257
00258 template <typename storageType>
00259 inline std::map<storageType, Markov::Edge<storageType>*>* Markov::Node<storageType>::Edges() {
00260     return &(this->edges);
00261 }
00262
00263 template <typename storageType>
00264 inline long int Markov::Node<storageType>::TotalEdgeWeights() {
00265     return this->total_edge_weights;
00266 }
00267

```

8.41 pch.cpp File Reference

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

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



8.42 MarkovModel/src/pch.cpp

```

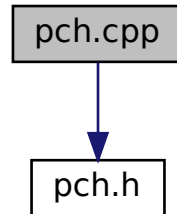
00001 // pch.cpp: source file corresponding to the pre-compiled header
00002
00003 #include "pch.h"
00004
00005 // When you are using pre-compiled headers, this source file is necessary for compilation to succeed.
00006

```

8.43 pch.cpp File Reference

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

Include dependency graph for UnitTests/pch.cpp:



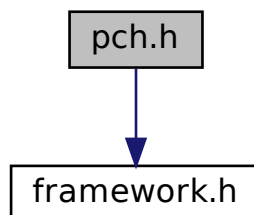
8.44 UnitTests/pch.cpp

```
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.
00006
00007
```

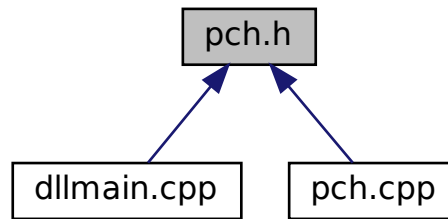
8.45 pch.h File Reference

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

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



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



8.46 MarkovModel/src/pch.h

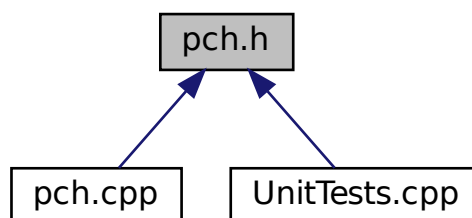
```

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
00014

```

8.47 pch.h File Reference

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



8.48 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
      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
00013
00014

```

8.49 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

8.50 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")

```

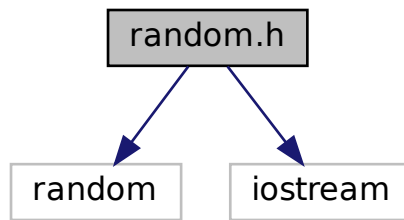
8.51 random.h File Reference

```

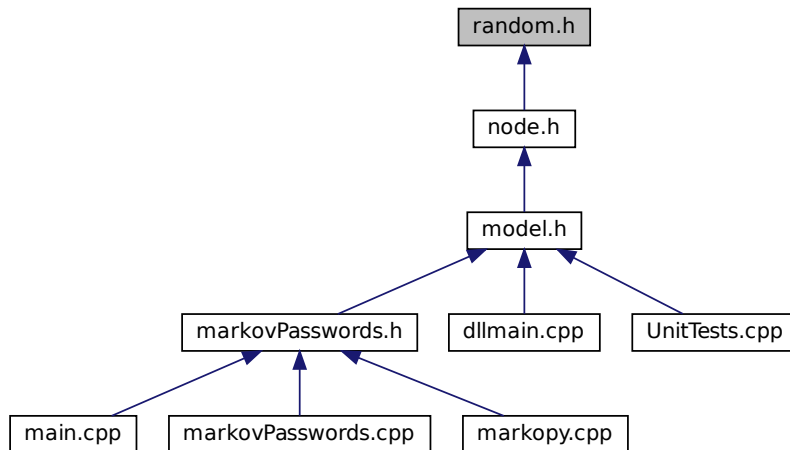
#include <random>
#include <iostream>

```

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.

8.52 random.h

```

00001
00002
00003 #include <random>
00004 #include <iostream>
00005
00009 namespace Markov::Random{
00010
00021     class RandomEngine{
00022     public:
00023         virtual inline unsigned long random() = 0;
00024     };
00025
00026
00027
00052     class DefaultRandomEngine : public RandomEngine{
00053     public:
00057         inline unsigned long random() {
00058             return this->distribution() (this->generator());
00059         }
00060     protected:
00061
00065         inline std::random_device& rd() {
00066             static std::random_device _rd;
00067             return _rd;
00068         }
00069
00073         inline std::default_random_engine& generator() {
00074             static std::default_random_engine _generator(rd() ());
00075             return _generator;
00076         }
00077
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
00116     class Marsaglia : public DefaultRandomEngine{
00117     public:
00118
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         inline unsigned long random(){
00130             unsigned long t;
00131             x ^= x << 16;
00132             x ^= x >> 5;
00133             x ^= x << 1;
00134
00135             t = x;
00136             x = y;
00137             y = z;
00138             z = t ^ x ^ y;
00139
00140             return z;
00141         }
00142
00143         unsigned long x;
00144         unsigned long y;
00145         unsigned long z;
00146     };
00147
00148
00149
00174     class Mersenne : public DefaultRandomEngine{
00175
00176     };
00177
00178
00179 };

```

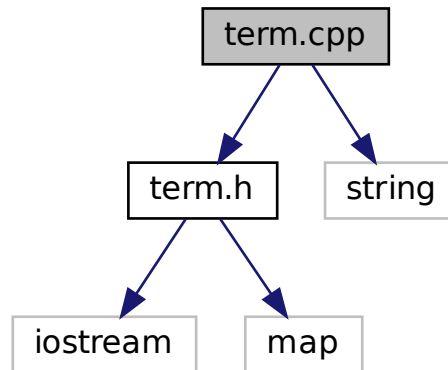
8.53 README.md File Reference

8.54 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)`

8.54.1 Function Documentation

8.54.1.1 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](#).

8.55 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[1;" << buf << "m";
00064     return os;
00065 }
00066
00067
00068
00069
00070 #endif
00071

```

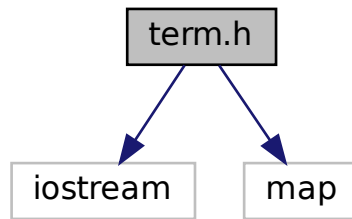
8.56 term.h File Reference

```

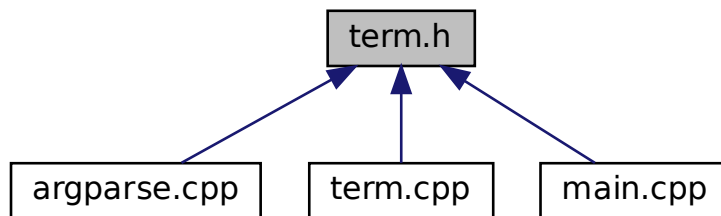
#include <iostream>
#include <map>

```


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`)

8.56.1 Macro Definition Documentation

8.56.1.1 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](#).

8.56.1.2 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](#).

8.56.1.3 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](#).

8.56.1.4 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](#).

8.57 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 << "+" <<
Markov::API::CLI::Terminal::color::RESET << "]" "
00011 #define TERM_INFO "[" << Markov::API::CLI::Terminal::color::BLUE << "+" <<
Markov::API::CLI::Terminal::color::RESET << "]" "
00012 #define TERM_WARN "[" << Markov::API::CLI::Terminal::color::YELLOW << "+" <<
Markov::API::CLI::Terminal::color::RESET << "]" "
00013 #define TERM_SUCC "[" << Markov::API::CLI::Terminal::color::GREEN << "+" <<
Markov::API::CLI::Terminal::color::RESET << "]" "
00014
00015 namespace Markov::API::CLI{
00016     class Terminal {
00017     public:
00018
00019         Terminal();
00020
00021         enum color { RESET, BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, LIGHTGRAY,
DARKGRAY, BROWN };
00022
00023         #ifdef _WIN32
00024             static HANDLE _stdout;
00025             static HANDLE _stderr;
00026             static std::map<Markov::API::CLI::Terminal::color, DWORD> colormap;
00027         #else
00028             static std::map<Markov::API::CLI::Terminal::color, int> colormap;
00029         #endif
00030     };
00031 }
```

```

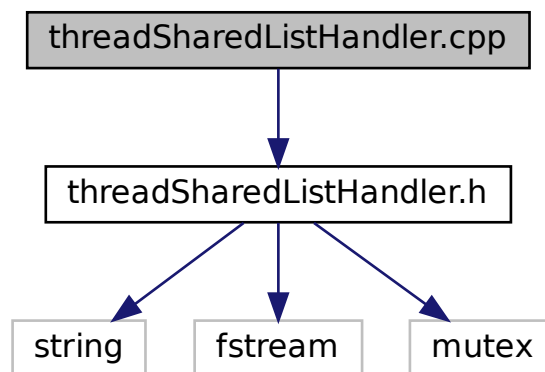
00034
00035
00036
00037     static std::ostream endl;
00038
00039
00040 };
00041
00042 std::ostream& operator<<(std::ostream& os, const Markov::API::CLI::Terminal::color& c);
00043
00044 }
00045
00046
00047

```

8.58 threadSharedListHandler.cpp File Reference

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

Include dependency graph for threadSharedListHandler.cpp:



8.59 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 }

```

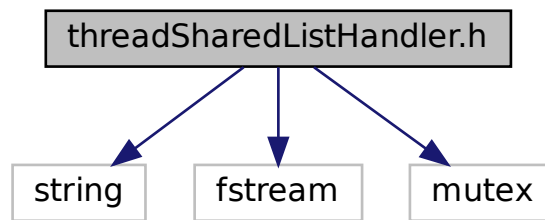
8.60 threadSharedListHandler.h File Reference

```

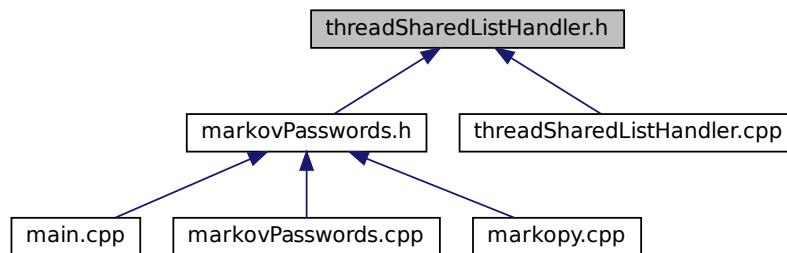
#include <string>
#include <fstream>
#include <mutex>

```

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.

8.61 threadSharedListHandler.h

```

00001 #include <string>
00002 #include <fstream>
00003 #include <mutex>
00004
00007 namespace Markov::API::Concurrency{
00008
00018 class ThreadSharedListHandler{
00019 public:
00071     ThreadSharedListHandler(const char* filename);
00072
  
```

```

00085     bool next(std::string* line);
00086
00087 private:
00088     std::ifstream listfile;
00089     std::mutex mlock;
00090 };
00091
00092 };

```

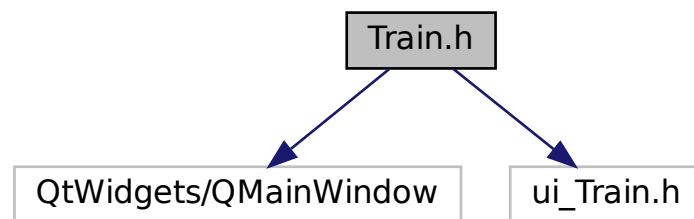
8.62 Train.h File Reference

```

#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

8.63 Train.h

```

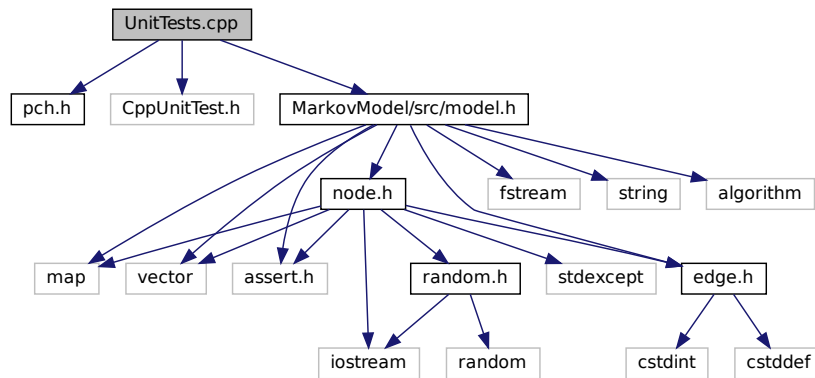
00001 #pragma once
00002 #include <QtWidgets/QMainWindow>
00003 #include "ui_Train.h"
00004
00005 namespace Markov::GUI{
00006
00009     class Train :public QMainWindow {
00010     Q_OBJECT
00011     public:
00012         Train(QWidget* parent = Q_NULLPTR);
00013
00014     private:
00015         Ui::Train ui;
00016
00017     public slots:
00018         void home();
00019         void train();
00020     };
00021 };
00022

```

8.64 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.

8.65 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
00010 namespace Testing {
00011
00014     namespace MVP {
00017         namespace MarkovModel
00018         {
00021             TEST_CLASS(Edge)
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,
00040 RightNode);
00041                     Assert::IsTrue(LeftNode == e->LeftNode());
00042                     Assert::IsTrue(RightNode == e->RightNode());
00043                     delete LeftNode;
00044                     delete RightNode;
00045                     delete e;
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,
00053 RightNode);
00054                     e->AdjustEdge(15);
00055                     Assert::AreEqual(15ull, e->EdgeWeight());
00056                     e->AdjustEdge(15);
00057                     Assert::AreEqual(30ull, e->EdgeWeight());
00058                     delete LeftNode;
00059                     delete RightNode;
00060                     delete e;
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,
00068 RightNode);
00069                     Assert::IsTrue(RightNode == e->TraverseNode());
00070                     delete LeftNode;
00071                     delete RightNode;
00072                     delete e;
00073                 }
00076                 TEST_METHOD(set_left_and_right) {
00077                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00078                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00079                     Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(LeftNode,
00080 RightNode);
00081                     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
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,
00099 RightNode);
00100                     e->AdjustEdge(15);
                     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
00111 TEST_CLASS(Node)
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
00214     TEST_METHOD(rand_next_choice_2) {
00215
00216         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00217         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00218         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00219         Markov::Edge<unsigned char>* e1 = src->Link(target1);
00220         Markov::Edge<unsigned char>* e2 = src->Link(target2);
00221         e2->AdjustEdge(1);
00222         e1->AdjustEdge((unsigned long)(1ull << 31));
00223         Markov::Node<unsigned char>* res = src->RandomNext();
00224         Assert::IsNotNull(res);
00225         Assert::IsTrue(res == target1);
00226         delete src;
00227         delete target1;
00228         delete e1;
00229         delete e2;
00230     }
00231
00232     TEST_METHOD(update_edges_count) {
00233
00234         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00235         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00236         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00237         Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00238         Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00239         e1->AdjustEdge(25);
00240         src->UpdateEdges(e1);
00241         e2->AdjustEdge(30);
00242         src->UpdateEdges(e2);
00243
00244         Assert::AreEqual((size_t)2, src->Edges()->size());
00245
00246         delete src;
00247         delete target1;
00248         delete e1;
00249         delete e2;
00250     }
00251
00252     TEST_METHOD(update_edges_total) {
00253
00254         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00255         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00256         Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00257         Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target1);
00258         e1->AdjustEdge(25);
00259         src->UpdateEdges(e1);
00260         e2->AdjustEdge(30);
00261         src->UpdateEdges(e2);
00262
00263         Assert::AreEqual(55ull, src->TotalEdgeWeights());
00264
00265         delete src;
00266         delete target1;
00267         delete e1;
00268         delete e2;
00269     }
00270
00271     TEST_METHOD(find_vertice) {
00272
00273         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00274         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00275         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00276         Markov::Edge<unsigned char>* res = NULL;
00277         src->Link(target1);
00278         src->Link(target2);
00279
00280         res = src->FindEdge('b');
00281         Assert::IsNotNull(res);
00282         Assert::AreEqual((unsigned char)'b', res->TraverseNode()->NodeValue());
00283         res = src->FindEdge('c');
00284         Assert::IsNotNull(res);
00285         Assert::AreEqual((unsigned char)'c', res->TraverseNode()->NodeValue());
00286     }

```

```

00299         delete src;
00300         delete target1;
00301         delete target2;
00302
00303     }
00304
00305     TEST_METHOD(find_vertice_without_any) {
00306
00307         auto _invalid_next = [] {
00308             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00309             Markov::Edge<unsigned char>* res = NULL;
00310
00311             res = src->FindEdge('b');
00312             Assert::IsNull(res);
00313
00314             delete src;
00315         };
00316
00317         //Assert::ExpectException<std::logic_error>(_invalid_next);
00318     }
00319
00320     TEST_METHOD(find_vertice_nonexistent) {
00321
00322         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00323         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00324         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00325         Markov::Edge<unsigned char>* res = NULL;
00326         src->Link(target1);
00327         src->Link(target2);
00328
00329         res = src->FindEdge('D');
00330         Assert::IsNull(res);
00331
00332         delete src;
00333         delete target1;
00334         delete target2;
00335     }
00336 };
00337
00338 TEST_CLASS(Model)
00339 {
00340 public:
00341     TEST_METHOD(model_constructor) {
00342         Markov::Model<unsigned char> m;
00343         Assert::AreEqual((unsigned char)'0', m.StarterNode()->NodeValue());
00344     }
00345
00346     TEST_METHOD(import_filename) {
00347         Markov::Model<unsigned char> m;
00348         Assert::IsTrue(m.Import("../MarkovPasswords/Models/2gram.mdl"));
00349     }
00350
00351     TEST_METHOD(export_filename) {
00352         Markov::Model<unsigned char> m;
00353         Assert::IsTrue(m.Export("../MarkovPasswords/Models/testcase.mdl"));
00354     }
00355
00356     TEST_METHOD(random_walk) {
00357         Markov::Model<unsigned char> m;
00358         Assert::IsTrue(m.Import("../models/finished.mdl"));
00359         Assert::IsNotNull(m.RandomWalk(1,12));
00360     }
00361 };
00362
00363 namespace MarkovPasswords
00364 {
00365     TEST_CLASS(ArgParser)
00366     {
00367     public:
00368         TEST_METHOD(generate_basic) {
00369             int argc = 8;
00370             char *argv[] = {"markov.exe", "generate", "-if", "model.mdl", "-of",
00371 "passwords.txt", "-n", "100"};
00372
00373             /*ProgramOptions *p = Argparse::parse(argc, argv);
00374             Assert::IsNotNull(p);
00375
00376             Assert::AreEqual(p->bImport, true);
00377             Assert::AreEqual(p->bExport, false);
00378             Assert::AreEqual(p->importname, "model.mdl");
00379             Assert::AreEqual(p->outputfilename, "passwords.txt");
00380             Assert::AreEqual(p->generateN, 100); */
00381         }
00382     };
00383 }

```

```

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         TEST_METHOD(train_basic_longname) {
00464             int argc = 4;
00465             char *argv[] = { "markov.exe", "train", "--exportfilename", "model.mdl" };
00466
00467             /*ProgramOptions* p = Argparse::parse(argc, argv);
00468             Assert::IsNotNull(p);
00469
00470             Assert::AreEqual(p->bImport, false);
00471             Assert::AreEqual(p->bExport, true);
00472             Assert::AreEqual(p->exportname, "model.mdl"); */
00473         }
00474
00475         }
00476     };
00477
00478 };
00479
00480 };
00481
00482 }
00483
00484 }
00485
00488 namespace MarkovModel {
00489
00492     TEST_CLASS(Edge)
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
00511 TEST_METHOD(except_integer_overflow) {
00512     auto _overflow_adjust = [] {
00513         Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00514         Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00515         Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00516             RightNode);
00517         e->AdjustEdge(~0ull);
00518         e->AdjustEdge(1);
00519         delete LeftNode;
00520         delete RightNode;
00521         delete e;
00522     };
00523     Assert::ExpectException<std::underflow_error>(_overflow_adjust);
00524 }
00525
00526 };
00527
00528 TEST_CLASS(Node)
00529 {
00530 public:
00531     TEST_METHOD(rand_next_u64) {
00532         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00533         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00534         Markov::Edge<unsigned char>* e = src->Link(target1);
00535         e->AdjustEdge((unsigned long)(1ull < 63));
00536         Markov::Node<unsigned char>* res = src->RandomNext();
00537         Assert::IsTrue(res == target1);
00538         delete src;
00539         delete target1;
00540         delete e;
00541     }
00542
00543     TEST_METHOD(rand_next_u64_max) {
00544         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00545         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00546         Markov::Edge<unsigned char>* e = src->Link(target1);
00547         e->AdjustEdge((0xffffFFF));
00548         Markov::Node<unsigned char>* res = src->RandomNext();
00549         Assert::IsTrue(res == target1);
00550         delete src;
00551         delete target1;
00552         delete e;
00553     }
00554
00555     TEST_METHOD(uninitialized_rand_next) {
00556         auto _invalid_next = [] {
00557             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00558             Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00559             Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(src, target1);
00560             Markov::Node<unsigned char>* res = src->RandomNext();
00561
00562             delete src;
00563             delete target1;
00564             delete e;
00565         };
00566         Assert::ExpectException<std::logic_error>(_invalid_next);
00567     }
00568
00569 };
00570
00571 TEST_CLASS(Model)
00572 {
00573 public:
00574     TEST_METHOD(functional_random_walk) {
00575         Markov::Model<unsigned char> m;
00576         Markov::Node<unsigned char>* starter = m.StarterNode();
00577         Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00578         Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
00579         Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00580         Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
00581         starter->Link(a)->AdjustEdge(1);
00582         a->Link(b)->AdjustEdge(1);
00583         b->Link(c)->AdjustEdge(1);
00584     }
00585 }

```

```
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
00630 namespace MarkovPasswords {
00631
00632 };
00633
00634 }
00635
```


Index

- [_left](#)
 - [Markov::Edge< NodeStorageType >, 61](#)
 - [_right](#)
 - [Markov::Edge< NodeStorageType >, 61](#)
 - [_value](#)
 - [Markov::Node< storageType >, 107](#)
 - [_weight](#)
 - [Markov::Edge< NodeStorageType >, 61](#)
- [about](#)
 - [Markov::GUI::about, 41](#)
 - [Markov::GUI::CLI, 48](#)
 - [Markov::GUI::menu, 85](#)
- [about.h, 119](#)
- [action](#)
 - [markopy_cli, 17](#)
- [AdjustEdge](#)
 - [Markov::API::MarkovPasswords, 65](#)
 - [Markov::Edge< NodeStorageType >, 56](#)
 - [Markov::Model< NodeStorageType >, 92](#)
- [alphabet](#)
 - [model_2gram, 22](#)
 - [random-model, 23](#)
- [Argparse](#)
 - [Markov::API::CLI::Argparse, 43](#)
- [argparse.cpp, 120](#)
- [argparse.h, 120, 122](#)
 - [BOOST_ALL_DYN_LINK, 122](#)
- [args](#)
 - [markopy_cli, 17](#)
- [bExport](#)
 - [Markov::API::CLI::_programOptions, 39](#)
- [bFailure](#)
 - [Markov::API::CLI::_programOptions, 39](#)
- [blImport](#)
 - [Markov::API::CLI::_programOptions, 39](#)
- [BLACK](#)
 - [Markov::API::CLI::Terminal, 112](#)
- [BLUE](#)
 - [Markov::API::CLI::Terminal, 112](#)
- [BOOST_ALL_DYN_LINK](#)
 - [argparse.h, 122](#)
- [BOOST_PYTHON_MODULE](#)
 - [Markov::Markopy, 21](#)
- [BOOST_PYTHON_STATIC_LIB](#)
 - [markopy.cpp, 133](#)
- [BROWN](#)
 - [Markov::API::CLI::Terminal, 112](#)
- [bulk](#)
 - [markopy_cli, 17](#)
- [CLI](#)
 - [Markov::GUI::CLI, 48](#)
- [CLI.h, 124, 125](#)
- [cli_generate](#)
 - [markopy_cli, 15](#)
- [cli_init](#)
 - [markopy_cli, 16](#)
- [cli_train](#)
 - [markopy_cli, 16](#)
- [color](#)
 - [Markov::API::CLI::Terminal, 112](#)
- [colormap](#)
 - [Markov::API::CLI::Terminal, 112](#)
- [corpus_list](#)
 - [markopy_cli, 17](#)
- [CYAN](#)
 - [Markov::API::CLI::Terminal, 112](#)
- [DARKGRAY](#)
 - [Markov::API::CLI::Terminal, 112](#)
- [datasetFile](#)
 - [Markov::API::MarkovPasswords, 74](#)
- [datasetname](#)
 - [Markov::API::CLI::_programOptions, 39](#)
- [default](#)
 - [markopy_cli, 17](#)
- [distribution](#)
 - [Markov::Random::DefaultRandomEngine, 51](#)
 - [Markov::Random::Marsaglia, 81](#)
 - [Markov::Random::Mersenne, 88](#)
- [dllmain.cpp, 125](#)
- [Edge](#)
 - [Markov::Edge< NodeStorageType >, 56](#)
- [edge.h, 126, 127](#)
- [Edges](#)
 - [Markov::API::MarkovPasswords, 66](#)
 - [Markov::Model< NodeStorageType >, 93](#)
 - [Markov::Node< storageType >, 102](#)
- [edges](#)
 - [Markov::API::MarkovPasswords, 74](#)
 - [Markov::Model< NodeStorageType >, 98](#)
 - [Markov::Node< storageType >, 108](#)
- [edgesV](#)
 - [Markov::Node< storageType >, 108](#)
- [EdgeWeight](#)
 - [Markov::Edge< NodeStorageType >, 57](#)
- [endl](#)

- Markov::API::CLI::Terminal, 113
- Export
 - Markov::API::MarkovPasswords, 66
 - Markov::Model< NodeStorageType >, 93, 94
- exportname
 - Markov::API::CLI::_programOptions, 39
- f
 - model_2gram, 22
 - random-model, 23
- FindEdge
 - Markov::Node< storageType >, 102
- framework.h, 128
 - WIN32_LEAN_AND_MEAN, 128
- Generate
 - Markov::API::MarkovPasswords, 67
- generateN
 - Markov::API::CLI::_programOptions, 39
- GenerateThread
 - Markov::API::MarkovPasswords, 68
- generator
 - Markov::Random::DefaultRandomEngine, 51
 - Markov::Random::Marsaglia, 82
 - Markov::Random::Mersenne, 88
- getProgramOptions
 - Markov::API::CLI::Argparse, 45
- GREEN
 - Markov::API::CLI::Terminal, 112
- help
 - markopy_cli, 17
 - Markov::API::CLI::Argparse, 45
- home
 - Markov::GUI::Train, 118
- Import
 - Markov::API::MarkovPasswords, 69
 - Markov::Model< NodeStorageType >, 94, 95
- importname
 - Markov::API::CLI::_programOptions, 39
- LeftNode
 - Markov::Edge< NodeStorageType >, 58
- LIGHTGRAY
 - Markov::API::CLI::Terminal, 112
- Link
 - Markov::Node< storageType >, 103, 104
- listfile
 - Markov::API::Concurrency::ThreadSharedListHandler, 116
- loadDataset
 - Markov::GUI::MarkovPasswordsGUI, 77
- MAGENTA
 - Markov::API::CLI::Terminal, 112
- main
 - src/main.cpp, 129
 - UI/src/main.cpp, 131
- main.cpp, 129–131
- markopy.cpp, 132, 133
 - BOOST_PYTHON_STATIC_LIB, 133
- markopy_cli, 15
 - action, 17
 - args, 17
 - bulk, 17
 - cli_generate, 15
 - cli_init, 16
 - cli_train, 16
 - corpus_list, 17
 - default, 17
 - help, 17
 - model, 17
 - model_base, 17
 - model_extension, 17
 - model_list, 18
 - output, 18
 - output_file_name, 18
 - output_forced, 18
 - parser, 18
 - True, 18
- markopy_cli.py, 133, 134
- Markov, 18
- Markov::API, 19
- Markov::API::CLI, 19
 - operator<<, 20
 - ProgramOptions, 20
- Markov::API::CLI::_programOptions, 37
 - bExport, 39
 - bFailure, 39
 - blmport, 39
 - datasetname, 39
 - exportname, 39
 - generateN, 39
 - importname, 39
 - outputfilename, 39
 - separator, 40
 - wordlistname, 40
- Markov::API::CLI::Argparse, 42
 - Argparse, 43
 - getProgramOptions, 45
 - help, 45
 - parse, 45
 - po, 47
 - setProgramOptions, 46
- Markov::API::CLI::Terminal, 110
 - BLACK, 112
 - BLUE, 112
 - BROWN, 112
 - color, 112
 - colormap, 112
 - CYAN, 112
 - DARKGRAY, 112
 - endl, 113
 - GREEN, 112
 - LIGHTGRAY, 112
 - MAGENTA, 112
 - RED, 112

- RESET, 112
- Terminal, 112
- WHITE, 112
- YELLOW, 112
- Markov::API::Concurrency, 20
- Markov::API::Concurrency::ThreadSharedListHandler, 113
 - listfile, 116
 - mlock, 116
 - next, 116
 - ThreadSharedListHandler, 115
- Markov::API::MarkovPasswords, 61
 - AdjustEdge, 65
 - datasetFile, 74
 - Edges, 66
 - edges, 74
 - Export, 66
 - Generate, 67
 - GenerateThread, 68
 - Import, 69
 - MarkovPasswords, 65
 - modelSavefile, 74
 - Nodes, 70
 - nodes, 74
 - OpenDatasetFile, 70
 - outputFile, 74
 - RandomWalk, 71
 - Save, 72
 - StarterNode, 72
 - starterNode, 75
 - Train, 72
 - TrainThread, 73
- Markov::Edge< NodeStorageType >, 53
 - _left, 61
 - _right, 61
 - _weight, 61
 - AdjustEdge, 56
 - Edge, 56
 - EdgeWeight, 57
 - LeftNode, 58
 - RightNode, 58
 - SetLeftEdge, 59
 - SetRightEdge, 59
 - TraverseNode, 60
- Markov::GUI, 20
- Markov::GUI::about, 40
 - about, 41
 - ui, 41
- Markov::GUI::CLI, 47
 - about, 48
 - CLI, 48
 - start, 49
 - statistics, 49
 - ui, 49
- Markov::GUI::MarkovPasswordsGUI, 75
 - loadDataset, 77
 - MarkovPasswordsGUI, 77
 - MarkovPasswordsGUI::benchmarkSelected, 77
 - MarkovPasswordsGUI::comparisonSelected, 77
 - MarkovPasswordsGUI::modelvisSelected, 77
 - MarkovPasswordsGUI::visualDebugSelected, 78
 - renderHTMLFile, 78
 - ui, 78
- Markov::GUI::menu, 84
 - about, 85
 - menu, 85
 - ui, 85
 - visualization, 85
- Markov::GUI::Train, 116
 - home, 118
 - Train, 118
 - train, 118
 - ui, 118
- Markov::Markopy, 21
 - BOOST_PYTHON_MODULE, 21
- Markov::Model< NodeStorageType >, 90
 - AdjustEdge, 92
 - Edges, 93
 - edges, 98
 - Export, 93, 94
 - Import, 94, 95
 - Model, 92
 - Nodes, 96
 - nodes, 98
 - RandomWalk, 96
 - StarterNode, 97
 - starterNode, 98
- Markov::Node< storageType >, 99
 - _value, 107
 - Edges, 102
 - edges, 108
 - edgesV, 108
 - FindEdge, 102
 - Link, 103, 104
 - Node, 101
 - NodeValue, 104
 - RandomNext, 105
 - total_edge_weights, 108
 - TotalEdgeWeights, 106
 - UpdateEdges, 106
 - UpdateTotalVerticeWeight, 107
- Markov::Random, 21
- Markov::Random::DefaultRandomEngine, 49
 - distribution, 51
 - generator, 51
 - random, 52
 - rd, 52
- Markov::Random::Marsaglia, 78
 - distribution, 81
 - generator, 82
 - Marsaglia, 81
 - random, 82
 - rd, 83
 - x, 83
 - y, 83
 - z, 83

- Markov::Random::Mersenne, 86
 - distribution, 88
 - generator, 88
 - random, 89
 - rd, 89
- Markov::Random::RandomEngine, 108
 - random, 110
- MarkovPasswords
 - Markov::API::MarkovPasswords, 65
- markovPasswords.cpp, 136
- markovPasswords.h, 138, 139
- MarkovPasswordsGUI
 - Markov::GUI::MarkovPasswordsGUI, 77
- MarkovPasswordsGUI.cpp, 140
- MarkovPasswordsGUI.h, 141, 142
- MarkovPasswordsGUI::benchmarkSelected
 - Markov::GUI::MarkovPasswordsGUI, 77
- MarkovPasswordsGUI::comparisonSelected
 - Markov::GUI::MarkovPasswordsGUI, 77
- MarkovPasswordsGUI::modelvisSelected
 - Markov::GUI::MarkovPasswordsGUI, 77
- MarkovPasswordsGUI::visualDebugSelected
 - Markov::GUI::MarkovPasswordsGUI, 78
- Marsaglia
 - Markov::Random::Marsaglia, 81
- menu
 - Markov::GUI::menu, 85
- menu.cpp, 142
- menu.h, 143, 144
- mlock
 - Markov::API::Concurrency::ThreadSharedListHandler, 116
- Model
 - Markov::Model< NodeStorageType >, 92
- model
 - markopy_cli, 17
- model.h, 144, 145
- model_2gram, 22
 - alphabet, 22
 - f, 22
- model_2gram.py, 148
- model_base
 - markopy_cli, 17
- model_extension
 - markopy_cli, 17
- model_list
 - markopy_cli, 18
- modelSavefile
 - Markov::API::MarkovPasswords, 74
- next
 - Markov::API::Concurrency::ThreadSharedListHandler, 116
- Node
 - Markov::Node< storageType >, 101
- node.h, 148, 149
- Nodes
 - Markov::API::MarkovPasswords, 70
 - Markov::Model< NodeStorageType >, 96
- nodes
 - Markov::API::MarkovPasswords, 74
 - Markov::Model< NodeStorageType >, 98
- NodeValue
 - Markov::Node< storageType >, 104
- OpenDatasetFile
 - Markov::API::MarkovPasswords, 70
- operator<<
 - Markov::API::CLI, 20
 - term.cpp, 157
- output
 - markopy_cli, 18
- output_file_name
 - markopy_cli, 18
- output_forced
 - markopy_cli, 18
- outputFile
 - Markov::API::MarkovPasswords, 74
- outputfilename
 - Markov::API::CLI::_programOptions, 39
- parse
 - Markov::API::CLI::Argparse, 45
- parser
 - markopy_cli, 18
- pch.cpp, 151, 152
- pch.h, 152, 153
- po
 - Markov::API::CLI::Argparse, 47
- ProgramOptions
 - Markov::API::CLI, 20
- random, 22
 - Markov::Random::DefaultRandomEngine, 52
 - Markov::Random::Marsaglia, 82
 - Markov::Random::Mersenne, 89
 - Markov::Random::RandomEngine, 110
- random-model, 23
 - alphabet, 23
 - f, 23
- random-model.py, 154
- random.h, 154, 156
- RandomNext
 - Markov::Node< storageType >, 105
- RandomWalk
 - Markov::API::MarkovPasswords, 71
 - Markov::Model< NodeStorageType >, 96
- rd
 - Markov::Random::DefaultRandomEngine, 52
 - Markov::Random::Marsaglia, 83
 - Markov::Random::Mersenne, 89
- README.md, 157
- RED
 - Markov::API::CLI::Terminal, 112
- renderHTMLFile
 - Markov::GUI::MarkovPasswordsGUI, 78
- RESET
 - Markov::API::CLI::Terminal, 112

- RightNode
 - Markov::Edge< NodeStorageType >, 58
- Save
 - Markov::API::MarkovPasswords, 72
- seperator
 - Markov::API::CLI::_programOptions, 40
- SetLeftEdge
 - Markov::Edge< NodeStorageType >, 59
- setProgramOptions
 - Markov::API::CLI::Argparse, 46
- SetRightEdge
 - Markov::Edge< NodeStorageType >, 59
- src/main.cpp
 - main, 129
- start
 - Markov::GUI::CLI, 49
- StarterNode
 - Markov::API::MarkovPasswords, 72
 - Markov::Model< NodeStorageType >, 97
- starterNode
 - Markov::API::MarkovPasswords, 75
 - Markov::Model< NodeStorageType >, 98
- statistics
 - Markov::GUI::CLI, 49
- term.cpp, 157
 - operator<=, 157
- term.h, 158, 160
 - TERM_FAIL, 160
 - TERM_INFO, 160
 - TERM_SUCC, 160
 - TERM_WARN, 160
- TERM_FAIL
 - term.h, 160
- TERM_INFO
 - term.h, 160
- TERM_SUCC
 - term.h, 160
- TERM_WARN
 - term.h, 160
- Terminal
 - Markov::API::CLI::Terminal, 112
- TEST_CLASS
 - Testing::MarkovModel, 24, 25
 - Testing::MVP::MarkovModel, 27, 29, 30
 - Testing::MVP::MarkovPasswords, 33
- Testing, 23
 - Testing::MarkovModel, 23
 - TEST_CLASS, 24, 25
- Testing::MarkovPasswords, 26
- Testing::MVP, 26
 - Testing::MVP::MarkovModel, 27
 - TEST_CLASS, 27, 29, 30
- Testing::MVP::MarkovPasswords, 33
 - TEST_CLASS, 33
- ThreadSharedListHandler
 - Markov::API::Concurrency::ThreadSharedListHandler, 115
- threadSharedListHandler.cpp, 161
- threadSharedListHandler.h, 161, 162
- total_edge_weights
 - Markov::Node< storageType >, 108
- TotalEdgeWeights
 - Markov::Node< storageType >, 106
- Train
 - Markov::API::MarkovPasswords, 72
 - Markov::GUI::Train, 118
- train
 - Markov::GUI::Train, 118
- Train.h, 163
- TrainThread
 - Markov::API::MarkovPasswords, 73
- TraverseNode
 - Markov::Edge< NodeStorageType >, 60
- True
 - markopy_cli, 18
- ui
 - Markov::GUI::about, 41
 - Markov::GUI::CLI, 49
 - Markov::GUI::MarkovPasswordsGUI, 78
 - Markov::GUI::menu, 85
 - Markov::GUI::Train, 118
- UI/src/main.cpp
 - main, 131
- UnitTests.cpp, 164, 165
- UpdateEdges
 - Markov::Node< storageType >, 106
- UpdateTotalVerticeWeight
 - Markov::Node< storageType >, 107
- visualization
 - Markov::GUI::menu, 85
- WHITE
 - Markov::API::CLI::Terminal, 112
- WIN32_LEAN_AND_MEAN
 - framework.h, 128
- wordlistname
 - Markov::API::CLI::_programOptions, 40
- x
 - Markov::Random::Marsaglia, 83
- y
 - Markov::Random::Marsaglia, 83
- YELLOW
 - Markov::API::CLI::Terminal, 112
- z
 - Markov::Random::Marsaglia, 83