Middle East Technical University Northern Cyprus Campus
Computer Engineering Program

CNG491 Computer Engineering Design I

Markopy Documentation

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

Supervised by
Assoc. Prof. Dr. Okan Topçu

0.4.1 Documentation

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">

</details>

1.1 About The Project

This project aims to generate wordlists using markov models.

1.1.1 Built With

- CPP, with dependecies: 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 PYTH←
ON_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 lboost is required but filenames are formatted as llibboost, or vice versa.
Do the same for python related library issues, run: `ls /usr/lib/x86_64-linux-gnu/ | grep python` to verify filename format is as required.
If not, you can modify the makefile, or create symlinks such as: `ln -s /usr/lib/x86_64-linux-gnu/libboost←_python38.so /usr/lib/x86_64-linux-gnu/boost_python38.so`

1.5.2 Windows

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

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

```
set INCLUDE=%INCLUDE%;C:\Program Files (x86)\Windows Kits\NETFXSDK\4.8\include\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 linugistic analysis ourselves, our Markov Model has highlighted that strings are more likely to start with a, b tends to follow a, and a is likely to be repeated in the string.

1.5.3 Contributing

Feel free to contribute.

1.5.4 Contact

Twitter - `@ahakcil`

Chapter 2

Deprecated List

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

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

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 markopy_cli Namespace Reference

Functions

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

Variables

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

7.1.1 Detailed Description

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

7.1.2 Function Documentation

7.1.2.1 cli_generate()

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

Definition at line 114 of file markopy_cli.py.

```
def cli_generate(model, wordlist, bulk=False):
    if not (wordlist or args.count):
        logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters. Exiting.")
        exit(2)

    if (bulk and os.path.isfile(wordlist)):
        logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
    model.Generate(int(args.count), wordlist, int(args.min), int(args.max), int(args.threads))


```

7.1.2.2 cli_init()

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

Definition at line 61 of file markopy_cli.py.

```
def cli_init(input_model):
    logging.VERBOSITY = 0
    if args.verbosity:
        logging.VERBOSITY = args.verbosity
        logging.pprint(f"Verbosity set to {args.verbosity}.", 2)

    logging.pprint("Initializing model.", 1)
    model = markopy.MarkovPasswords()
    logging.pprint("Model initialized.", 2)

    logging.pprint("Importing model file.", 1)

    if (not os.path.isfile(input_model)):
        logging.pprint(f"Model file at {input_model} not found. Check the file path, or working
directory")
        exit(1)

    model.Import(input_model)
    logging.pprint("Model imported successfully.", 2)
    return model


```

7.1.2.3 cli_train()

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

Definition at line 82 of file markopy_cli.py.

```
def cli_train(model, dataset, seperator, output, output_forced=False, bulk=False):
    if not (dataset and seperator and (output or not output_forced)):
        logging.pprint(
            f"Training mode requires -d/--dataset{', -o/--output' if output_forced else ''} and
-s/--seperator parameters. Exiting.")
        exit(2)

    if (not bulk and not os.path.isfile(dataset)):
        logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
        exit(3)

    if (output and os.path.isfile(output)):
        logging.pprint(f"{output} exists and will be overwritten.", 1)

    if (seperator == '\\t'):
        logging.pprint("Escaping seperator.", 3)
        seperator = '\t'

    if (len(seperator) != 1):
        logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not accepted.')
        exit(4)

    logging.pprint(f'Starting training.', 3)
    model.Train(dataset, seperator, int(args.threads))
    logging.pprint(f'Training completed.', 2)

    if (output):
        logging.pprint(f'Exporting model to {output}', 2)
        model.Export(output)
    else:
        logging.pprint(f'Model will not be exported.', 1)


```

7.1.3 Variable Documentation

7.1.3.1 action

`markopy_cli.action`

Definition at line 49 of file markopy_cli.py.

7.1.3.2 args

`markopy_cli.args = parser.parse_args()`

Definition at line 58 of file markopy_cli.py.

7.1.3.3 bulk

`markopy_cli.bulk`

Definition at line 139 of file markopy_cli.py.

7.1.3.4 corpus_list

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

Definition at line 130 of file markopy_cli.py.

7.1.3.5 default

`markopy_cli.default`

Definition at line 41 of file markopy_cli.py.

7.1.3.6 help

`markopy_cli.help`

Definition at line 27 of file markopy_cli.py.

7.1.3.7 model

`def markopy_cli.model = cli_init(args.input)`

Definition at line 132 of file markopy_cli.py.

7.1.3.8 model_base

`markopy_cli.model_base = input`

Definition at line 153 of file markopy_cli.py.

7.1.3.9 model_extension

`markopy_cli.model_extension = ""`
Definition at line 135 of file markopy_cli.py.

7.1.3.10 model_list

`markopy_cli.model_list = os.listdir(args.input)`
Definition at line 147 of file markopy_cli.py.

7.1.3.11 output

`markopy_cli.output`
Definition at line 167 of file markopy_cli.py.

7.1.3.12 output_file_name

`markopy_cli.output_file_name = corpus`
Definition at line 134 of file markopy_cli.py.

7.1.3.13 output_forced

`markopy_cli.output_forced`
Definition at line 139 of file markopy_cli.py.

7.1.3.14 parser

`markopy_cli.parser`
Initial value:
```
00001 =  argparse.ArgumentParser(description="Python wrapper for MarkovPasswords.",
00002                                    epilog=f, formatter_class=argparse.RawTextHelpFormatter)
```
Definition at line 12 of file markopy_cli.py.

7.1.3.15 True

`markopy_cli.True`
Definition at line 139 of file markopy_cli.py.

7.2 Markov Namespace Reference

Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

Namespaces

- API

  Namespace for the MarkovPasswords API.
- GUI

  namespace for MarkovPasswords API GUI wrapper
- Markopy
- Random

  Objects related to RNG.

Classes

- class Edge

  Edge class used to link nodes in the model together.
- class Model

  class for the final Markov Model, constructed from nodes and edges.
- class Node

  A node class that for the vertices of model. Connected with eachother using Edge.

7.2.1 Detailed Description

Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

7.3 Markov::API Namespace Reference

Namespace for the MarkovPasswords API.

Namespaces

- CLI

  Structure to hold parsed cli arguements.
- Concurrency

  Namespace for Concurrency related classes.
- CUDA

  Namespace for objects requiring CUDA libraries.

Classes

- class MarkovPasswords

  Markov::Model with char represented nodes.
- class ModelMatrix

  Class to flatten and reduce Markov::Model to a Matrix.

7.3.1 Detailed Description

Namespace for the MarkovPasswords API.

7.4 Markov::API::CLI Namespace Reference

Structure to hold parsed cli arguements.

Classes

- struct _programOptions

  Structure to hold parsed cli arguements.
- class Argparse

  Parse command line arguements.
- class Terminal

  pretty colors for Terminal. Windows Only.

Typedefs

- typedef struct Markov::API::CLI::_programOptions ProgramOptions

  Structure to hold parsed cli arguements.

Functions

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

7.4.1 Detailed Description

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

7.4.2 Typedef Documentation

7.4.2.1 ProgramOptions

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

Structure to hold parsed cli arguements.

7.4.3 Function Documentation

7.4.3.1 operator<<()

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

overload for std::cout.
Definition at line 60 of file term.cpp.

```
                                                                  {
    char buf[6];
    sprintf(buf,"%d",Terminal::colormap.find(c)->second);
    os « "\e[1;" « buf « "m";
    return os;
}
```

References Markov::API::CLI::Terminal::colormap.

7.5 Markov::API::Concurrency Namespace Reference

Namespace for Concurrency related classes.

Classes

- class ThreadSharedListHandler

    Simple class for managing shared access to file.

7.5.1 Detailed Description

Namespace for Concurrency related classes.

7.6 Markov::API::CUDA Namespace Reference

Namespace for objects requiring CUDA libraries.

Classes

- class CUDADeviceController

    Controller class for CUDA device.

- class CUDAModelMatrix

Extension of Markov::API::ModelMatrix which is modified to run on GPU devices.

7.6.1 Detailed Description

Namespace for objects requiring CUDA libraries.

7.7 Markov::GUI Namespace Reference

namespace for MarkovPasswords API GUI wrapper

Classes

- class about

  QT Class for about page.
- class CLI

  QT CLI Class.
- class MarkovPasswordsGUI

  Reporting UI.
- class menu

  QT Menu class.
- class Train

  QT Training page class.

7.7.1 Detailed Description

namespace for MarkovPasswords API GUI wrapper

7.8 Markov::Markopy Namespace Reference

Functions

- BOOST_PYTHON_MODULE (markopy)

7.8.1 Function Documentation

7.8.1.1 BOOST_PYTHON_MODULE()

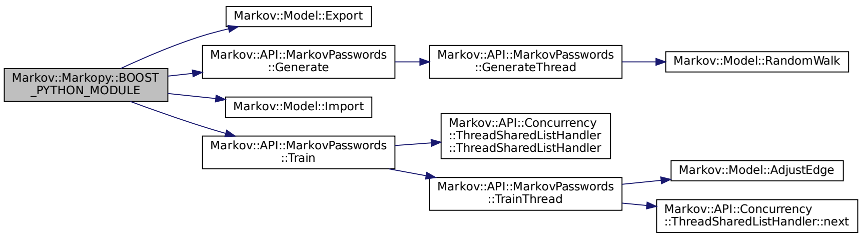
```
Markov::Markopy::BOOST_PYTHON_MODULE (
              markopy  )
```

Definition at line 11 of file markopy.cpp.

```
    {
        bool (Markov::API::MarkovPasswords::*Import)(const char*) = &Markov::Model<char>::Import;
        bool (Markov::API::MarkovPasswords::*Export)(const char*) = &Markov::Model<char>::Export;
        class_<Markov::API::MarkovPasswords>("MarkovPasswords", init<>())
            .def(init<>())
            .def("Train", &Markov::API::MarkovPasswords::Train,
            "Train the model\n"
            "\n"
            ":param datasetFileName: Ifstream* to the dataset. If null, use class member\n"
            ":param delimiter: a character, same as the delimiter in dataset content\n"
            ":param threads: number of OS threads to spawn\n")
            .def("Generate", &Markov::API::MarkovPasswords::Generate,
            "Generate passwords from a trained model.\n"
            ":param n: Ifstream* to the dataset. If null, use class member\n"
            ":param wordlistFileName: a character, same as the delimiter in dataset content\n"
            ":param minLen: number of OS threads to spawn\n"
            ":param maxLen: Ifstream* to the dataset. If null, use class member\n"
            ":param threads: a character, same as the delimiter in dataset content\n"
            ":param threads: number of OS threads to spawn\n")
            .def("Import", Import, "Import a model file.")
            .def("Export", Export, "Export a model to file.")
        ;
    };
```

References Markov::Model< NodeStorageType >::Export(), Markov::API::MarkovPasswords::Generate(), Markov::Model< NodeStor
and Markov::API::MarkovPasswords::Train().

Here is the call graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::Model::Export
Markov::API::MarkovPasswords ::Generate
Markov::API::MarkovPasswords ::GenerateThread
Markov::Model::RandomWalk
Markov::Model::Import
Markov::API::MarkovPasswords ::Train
Markov::API::Concurrency ::ThreadSharedListHandler ::ThreadSharedListHandler
Markov::API::MarkovPasswords ::TrainThread
Markov::Model::AdjustEdge
Markov::API::Concurrency ::ThreadSharedListHandler::next

7.9 Markov::Random Namespace Reference

Objects related to RNG.

Classes

- class DefaultRandomEngine

  Implementation using Random.h default random engine.
- class Marsaglia

  Implementation of Marsaglia Random Engine.
- class Mersenne

  Implementation of Mersenne Twister Engine.
- class RandomEngine

  An abstract class for Random Engine.

7.9.1 Detailed Description

Objects related to RNG.

7.10 model_2gram Namespace Reference

Variables

- alphabet = string.printable

  password alphabet
- f = open('../../models/2gram.mdl', "wb")

  output file handle

7.10.1 Detailed Description

```
python script for generating a 2gram model
```

7.10.2 Variable Documentation

7.10.2.1 alphabet

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

7.10.2.2 f

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

7.11 random Namespace Reference

7.11.1 Detailed Description

-model

```
  python script for generating a 2gram model
```

7.12 random-model Namespace Reference

Variables

- alphabet = string.printable

    password alphabet

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

    output file handle

7.12.1 Variable Documentation

7.12.1.1 alphabet

`random-model.alphabet = string.printable`
password alphabet
Definition at line 10 of file random-model.py.

7.12.1.2 f

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

7.13 Testing Namespace Reference

Namespace for Microsoft Native Unit Testing Classes.

Namespaces

- MarkovModel

  Testing namespace for MarkovModel.
- MarkovPasswords

  Testing namespace for MarkovPasswords.
- MVP

  Testing Namespace for Minimal Viable Product.

7.13.1 Detailed Description

Namespace for Microsoft Native Unit Testing Classes.

7.14 Testing::MarkovModel Namespace Reference

Testing namespace for MarkovModel.

Functions

- TEST_CLASS (Edge)

  Test class for rest of Edge cases.
- TEST_CLASS (Node)

  Test class for rest of Node cases.
- TEST_CLASS (Model)

  Test class for rest of model cases.

7.14.1 Detailed Description

Testing namespace for MarkovModel.

7.14.2 Function Documentation

7.14.2.1 TEST_CLASS() [1/3]

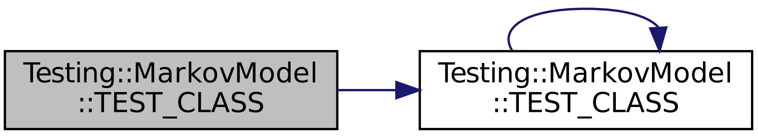
```
Testing::MarkovModel::TEST_CLASS (
              Edge  )
```

Test class for rest of Edge cases.
send exception on integer underflow
test integer overflows
Definition at line 492 of file UnitTests.cpp.

```
        {
        public:
            TEST_METHOD(except_integer_underflow) {
                auto _underflow_adjust = [] {
                    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 = new Markov::Edge<unsigned char>(LeftNode,
       RightNode);
                    e->AdjustEdge(15);
                    e->AdjustEdge(-30);
                    delete LeftNode;
                    delete RightNode;
                    delete e;
                };
                Assert::ExpectException<std::underflow_error>(_underflow_adjust);
            }

            TEST_METHOD(except_integer_overflow) {
                auto _overflow_adjust = [] {
                    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 = new Markov::Edge<unsigned char>(LeftNode,
       RightNode);
                    e->AdjustEdge(~0ull);
                    e->AdjustEdge(1);
                    delete LeftNode;
                    delete RightNode;
                    delete e;
                };
                Assert::ExpectException<std::underflow_error>(_overflow_adjust);
            }
        };
```

References TEST_CLASS().

Here is the call graph for this function:



7.14.2.2 TEST_CLASS() [2/3]

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

Test class for rest of model cases.

Definition at line 589 of file UnitTests.cpp.

```
        {
        public:
            TEST_METHOD(functional_random_walk) {
                Markov::Model<unsigned char> m;
                Markov::Node<unsigned char>* starter = m.StarterNode();
                Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
                Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
                Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
                Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
                starter->Link(a)->AdjustEdge(1);
                a->Link(b)->AdjustEdge(1);
                b->Link(c)->AdjustEdge(1);
                c->Link(end)->AdjustEdge(1);

                char* res = (char*)m.RandomWalk(1,12);
                Assert::IsFalse(strcmp(res, "abc"));
            }
            TEST_METHOD(functionoal_random_walk_without_any) {
                Markov::Model<unsigned char> m;
                Markov::Node<unsigned char>* starter = m.StarterNode();
                Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
                Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
                Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
                Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
                Markov::Edge<unsigned char>* res = NULL;
                starter->Link(a)->AdjustEdge(1);
                a->Link(b)->AdjustEdge(1);
                b->Link(c)->AdjustEdge(1);
                c->Link(end)->AdjustEdge(1);

                res = starter->FindEdge('D');
                Assert::IsNull(res);

            }
        };
```

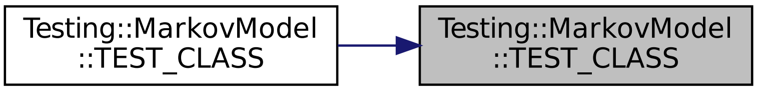
References TEST_CLASS().

Referenced by TEST_CLASS().

Here is the call graph for this function:

Testing::MarkovModel
::TEST_CLASS

Here is the caller graph for this function:



7.14.2.3 TEST_CLASS() [3/3]

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

Test class for rest of Node cases.

test RandomNext with 64 bit high values

test RandomNext with 64 bit high values

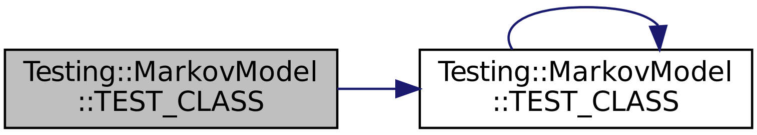
randomNext when no edges are present

Definition at line 530 of file UnitTests.cpp.

```
        {
        public:

            TEST_METHOD(rand_next_u64) {

                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>* e = src->Link(target1);
                e->AdjustEdge((unsigned long)(1ull « 63));
                Markov::Node<unsigned char>* res = src->RandomNext();
                Assert::IsTrue(res == target1);
                delete src;
                delete target1;
                delete e;

            }

            TEST_METHOD(rand_next_u64_max) {

                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>* e = src->Link(target1);
                e->AdjustEdge((0xffffFFFF));
                Markov::Node<unsigned char>* res = src->RandomNext();
                Assert::IsTrue(res == target1);
                delete src;
                delete target1;
                delete e;

            }

            TEST_METHOD(uninitialized_rand_next) {

                auto _invalid_next = [] {
                    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>* e = new Markov::Edge<unsigned char>(src, target1);
                    Markov::Node<unsigned char>* res = src->RandomNext();

                    delete src;
                    delete target1;
                    delete e;
                };

                Assert::ExpectException<std::logic_error>(_invalid_next);
            }


        };
```

References TEST_CLASS().

Here is the call graph for this function:


Testing::MarkovModel ::TEST_CLASS
Testing::MarkovModel ::TEST_CLASS

7.15 Testing::MarkovPasswords Namespace Reference

Testing namespace for MarkovPasswords.

7.15.1 Detailed Description

Testing namespace for MarkovPasswords.

7.16 Testing::MVP Namespace Reference

Testing Namespace for Minimal Viable Product.

Namespaces

- MarkovModel

  Testing Namespace for MVP MarkovModel.
- MarkovPasswords

  Testing namespace for MVP MarkovPasswords.

7.16.1 Detailed Description

Testing Namespace for Minimal Viable Product.

7.17 Testing::MVP::MarkovModel Namespace Reference

Testing Namespace for MVP MarkovModel.

Functions

- TEST_CLASS (Edge)

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

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

  Test class for minimal viable Model.

7.17.1 Detailed Description

Testing Namespace for MVP MarkovModel.

7.17.2 Function Documentation

7.17.2.1 TEST_CLASS() [1/3]

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

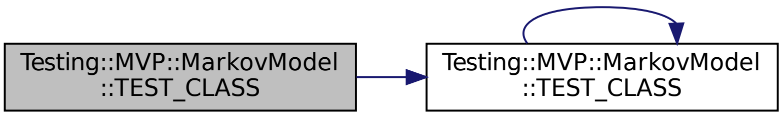
Test class for minimal viable Edge.
test default constructor
test linked constructor with two nodes
test AdjustEdge function
test TraverseNode returning RightNode
test LeftNode/RightNode setter
test negative adjustments
Definition at line 21 of file UnitTests.cpp.

```
          {
          public:

              TEST_METHOD(default_constructor) {
                  Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>;
                  Assert::IsNull(e->LeftNode());
                  Assert::IsNull(e->RightNode());
                  delete e;
              }

              TEST_METHOD(linked_constructor) {
                  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 = new Markov::Edge<unsigned char>(LeftNode,
   RightNode);
                  Assert::IsTrue(LeftNode == e->LeftNode());
                  Assert::IsTrue(RightNode == e->RightNode());
                  delete LeftNode;
                  delete RightNode;
                  delete e;
              }

              TEST_METHOD(AdjustEdge) {
                  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 = new Markov::Edge<unsigned char>(LeftNode,
   RightNode);
                  e->AdjustEdge(15);
                  Assert::AreEqual(15ull, e->EdgeWeight());
                  e->AdjustEdge(15);
                  Assert::AreEqual(30ull, e->EdgeWeight());
                  delete LeftNode;
                  delete RightNode;
                  delete e;
              }

              TEST_METHOD(TraverseNode) {
                  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 = new Markov::Edge<unsigned char>(LeftNode,
   RightNode);
                  Assert::IsTrue(RightNode == e->TraverseNode());
                  delete LeftNode;
                  delete RightNode;
```

```
                        delete e;
                    }

                    TEST_METHOD(set_left_and_right) {
                        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>* e1 = new Markov::Edge<unsigned char>(LeftNode,
       RightNode);

                        Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>;
                        e2->SetLeftEdge(LeftNode);
                        e2->SetRightEdge(RightNode);

                        Assert::IsTrue(e1->LeftNode() == e2->LeftNode());
                        Assert::IsTrue(e1->RightNode() == e2->RightNode());
                        delete LeftNode;
                        delete RightNode;
                        delete e1;
                        delete e2;
                    }

                    TEST_METHOD(negative_adjust) {
                        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 = new Markov::Edge<unsigned char>(LeftNode,
       RightNode);
                        e->AdjustEdge(15);
                        Assert::AreEqual(15ull, e->EdgeWeight());
                        e->AdjustEdge(-15);
                        Assert::AreEqual(0ull, e->EdgeWeight());
                        delete LeftNode;
                        delete RightNode;
                        delete e;
                    }
                };
```

References TEST_CLASS().

Here is the call graph for this function:


Testing::MVP::MarkovModel ::TEST_CLASS
Testing::MVP::MarkovModel ::TEST_CLASS

7.17.2.2 TEST_CLASS() [2/3]

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

Test class for minimal viable Model.

test model constructor for starter node

test import

test export

test random walk

Definition at line 347 of file UnitTests.cpp.

```
                {
                public:
                    TEST_METHOD(model_constructor) {
                        Markov::Model<unsigned char> m;
                        Assert::AreEqual((unsigned char)'\0', m.StarterNode()->NodeValue());
                    }

                    TEST_METHOD(import_filename) {
                        Markov::Model<unsigned char> m;
                        Assert::IsTrue(m.Import("../MarkovPasswords/Models/2gram.mdl"));
                    }

                    TEST_METHOD(export_filename) {
                        Markov::Model<unsigned char> m;
                        Assert::IsTrue(m.Export("../MarkovPasswords/Models/testcase.mdl"));
                    }

                    TEST_METHOD(random_walk) {
                        Markov::Model<unsigned char> m;
                        Assert::IsTrue(m.Import("../../models/finished.mdl"));
                        Assert::IsNotNull(m.RandomWalk(1,12));
                    }
                };
```

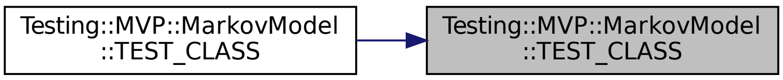
References TEST_CLASS().

Referenced by TEST_CLASS().

Here is the call graph for this function:

Testing::MVP::MarkovModel ::TEST_CLASS

Here is the caller graph for this function:


Testing::MVP::MarkovModel ::TEST_CLASS
Testing::MVP::MarkovModel ::TEST_CLASS

7.17.2.3 TEST_CLASS() [3/3]

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

Test class for minimal viable Node.

test default constructor

test custom constructor with unsigned char

test link function

test link function

test RandomNext with low values

test RandomNext with 32 bit high values

random next on a node with no follow-ups

random next on a node with no follow-ups

test updateEdges

test updateEdges

test FindVertice

test FindVertice

test FindVertice

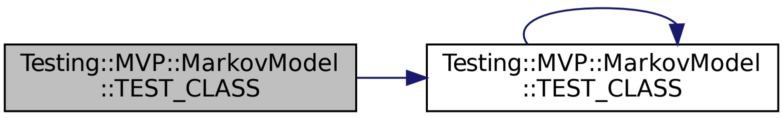
Definition at line 111 of file UnitTests.cpp.

```
            {
            public:

                TEST_METHOD(default_constructor) {
                    Markov::Node<unsigned char>* n = new Markov::Node<unsigned char>();
                    Assert::AreEqual((unsigned char)0, n->NodeValue());
                    delete n;
                }

                TEST_METHOD(uchar_constructor) {
                    Markov::Node<unsigned char>* n = NULL;
                    unsigned char test_cases[] = { 'c', 0x00, 0xff, -32 };
                    for (unsigned char tcase : test_cases) {
                        n = new Markov::Node<unsigned char>(tcase);
                        Assert::AreEqual(tcase, n->NodeValue());
                        delete n;
                    }
                }

                TEST_METHOD(link_left) {
                    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);
                    delete LeftNode;
                    delete RightNode;
                    delete e;
                }

                TEST_METHOD(link_right) {
                    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 = new Markov::Edge<unsigned char>(NULL, RightNode);
                    LeftNode->Link(e);
                    Assert::IsTrue(LeftNode == e->LeftNode());
                    Assert::IsTrue(RightNode == e->RightNode());
                    delete LeftNode;
                    delete RightNode;
                    delete e;
                }

                TEST_METHOD(rand_next_low) {

                    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>* e = src->Link(target1);
                    e->AdjustEdge(15);
                    Markov::Node<unsigned char>* res = src->RandomNext();
                    Assert::IsTrue(res == target1);
                    delete src;
                    delete target1;
                    delete e;

                }

                TEST_METHOD(rand_next_u32) {

                    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>* e = src->Link(target1);
                    e->AdjustEdge(1 « 31);
                    Markov::Node<unsigned char>* res = src->RandomNext();
                    Assert::IsTrue(res == target1);
                    delete src;
                    delete target1;
                    delete e;

                }

                TEST_METHOD(rand_next_choice_1) {

                    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 = src->Link(target1);
                    Markov::Edge<unsigned char>* e2 = src->Link(target2);
                    e1->AdjustEdge(1);
                    e2->AdjustEdge((unsigned long)(1ull « 31));
                    Markov::Node<unsigned char>* res = src->RandomNext();
                    Assert::IsNotNull(res);
                    Assert::IsTrue(res == target2);
                    delete src;
                    delete target1;
                    delete e1;
                    delete e2;
                }

                TEST_METHOD(rand_next_choice_2) {

                    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 = src->Link(target1);
                    Markov::Edge<unsigned char>* e2 = src->Link(target2);
                    e2->AdjustEdge(1);
                    e1->AdjustEdge((unsigned long)(1ull « 31));
                    Markov::Node<unsigned char>* res = src->RandomNext();
                    Assert::IsNotNull(res);
                    Assert::IsTrue(res == target1);
                    delete src;
                    delete target1;
                    delete e1;
                    delete e2;
                }


                TEST_METHOD(update_edges_count) {

                    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);

                    Assert::AreEqual((size_t)2, src->Edges()->size());

                    delete src;
                    delete target1;
                    delete e1;
                    delete e2;

                }

                TEST_METHOD(update_edges_total) {

                    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);
                    Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target1);
                    e1->AdjustEdge(25);
                    src->UpdateEdges(e1);
                    e2->AdjustEdge(30);
                    src->UpdateEdges(e2);

                    Assert::AreEqual(55ull, src->TotalEdgeWeights());

                    delete src;
                    delete target1;
                    delete e1;
                    delete e2;
                }


                TEST_METHOD(find_vertice) {

                    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');
                    Assert::IsNotNull(res);
                    Assert::AreEqual((unsigned char)'b', res->TraverseNode()->NodeValue());
                    res = src->FindEdge('c');
                    Assert::IsNotNull(res);
                    Assert::AreEqual((unsigned char)'c', res->TraverseNode()->NodeValue());

                    delete src;
                    delete target1;
                    delete target2;


                }


                TEST_METHOD(find_vertice_without_any) {

                    auto _invalid_next = [] {
                        Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
                        Markov::Edge<unsigned char>* res = NULL;

                        res = src->FindEdge('b');
```

```
                        Assert::IsNull(res);

                        delete src;
                    };

                    //Assert::ExpectException<std::logic_error>(_invalid_next);
                }

                TEST_METHOD(find_vertice_nonexistent) {

                    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('D');
                    Assert::IsNull(res);

                    delete src;
                    delete target1;
                    delete target2;

                }
            };
```

References TEST_CLASS().

Here is the call graph for this function:


Testing::MVP::MarkovModel ::TEST_CLASS
Testing::MVP::MarkovModel ::TEST_CLASS

7.18 Testing::MVP::MarkovPasswords Namespace Reference

Testing namespace for MVP MarkovPasswords.

Functions

- TEST_CLASS (ArgParser)

  Test Class for Argparse class.

7.18.1 Detailed Description

Testing namespace for MVP MarkovPasswords.

7.18.2 Function Documentation

7.18.2.1 TEST_CLASS()

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

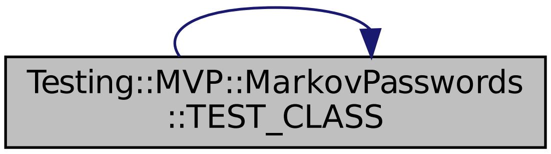
Test Class for Argparse class.
test basic generate
test basic generate reordered params
test basic generate param longnames
test basic generate
test basic generate

test basic generate

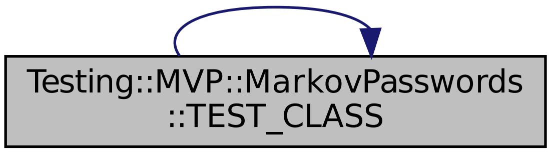
Definition at line 387 of file UnitTests.cpp.

```
        {
        public:
            TEST_METHOD(generate_basic) {
                int argc = 8;
                char *argv[] = {"markov.exe", "generate", "-if", "model.mdl", "-of", "passwords.txt", "-n", "100"};

                /*ProgramOptions *p = Argparse::parse(argc, argv);
                Assert::IsNotNull(p);

                Assert::AreEqual(p->bImport, true);
                Assert::AreEqual(p->bExport, false);
                Assert::AreEqual(p->importname, "model.mdl");
                Assert::AreEqual(p->outputfilename, "passwords.txt");
                Assert::AreEqual(p->generateN, 100); */

            }

            TEST_METHOD(generate_basic_reorder) {
                int argc = 8;
                char *argv[] = { "markov.exe", "generate", "-n", "100", "-if", "model.mdl", "-of", "passwords.txt" };

                /*ProgramOptions* p = Argparse::parse(argc, argv);
                Assert::IsNotNull(p);

                Assert::AreEqual(p->bImport, true);
                Assert::AreEqual(p->bExport, false);
                Assert::AreEqual(p->importname, "model.mdl");
                Assert::AreEqual(p->outputfilename, "passwords.txt");
                Assert::AreEqual(p->generateN, 100);*/
            }

            TEST_METHOD(generate_basic_longname) {
                int argc = 8;
                char *argv[] = { "markov.exe", "generate", "-n", "100", "--inputfilename", "model.mdl", "--outputfilename", "passwords.txt" };

                /*ProgramOptions* p = Argparse::parse(argc, argv);
                Assert::IsNotNull(p);

                Assert::AreEqual(p->bImport, true);
                Assert::AreEqual(p->bExport, false);
                Assert::AreEqual(p->importname, "model.mdl");
                Assert::AreEqual(p->outputfilename, "passwords.txt");
                Assert::AreEqual(p->generateN, 100); */
            }

            TEST_METHOD(generate_fail_badmethod) {
                int argc = 8;
                char *argv[] = { "markov.exe", "junk", "-n", "100", "--inputfilename", "model.mdl", "--outputfilename", "passwords.txt" };

                /*ProgramOptions* p = Argparse::parse(argc, argv);
                Assert::IsNull(p); */
            }

            TEST_METHOD(train_basic) {
                int argc = 4;
                char *argv[] = { "markov.exe", "train", "-ef", "model.mdl" };

                /*ProgramOptions* p = Argparse::parse(argc, argv);
                Assert::IsNotNull(p);

                Assert::AreEqual(p->bImport, false);
                Assert::AreEqual(p->bExport, true);
                Assert::AreEqual(p->exportname, "model.mdl"); */

            }

            TEST_METHOD(train_basic_longname) {
                int argc = 4;
                char *argv[] = { "markov.exe", "train", "--exportfilename", "model.mdl" };

                /*ProgramOptions* p = Argparse::parse(argc, argv);
                Assert::IsNotNull(p);

                Assert::AreEqual(p->bImport, false);
                Assert::AreEqual(p->bExport, true);
                Assert::AreEqual(p->exportname, "model.mdl"); */
            }



        };
```

References TEST_CLASS().
Referenced by TEST_CLASS().
Here is the call graph for this function:



Here is the caller graph for this function:



Chapter 8

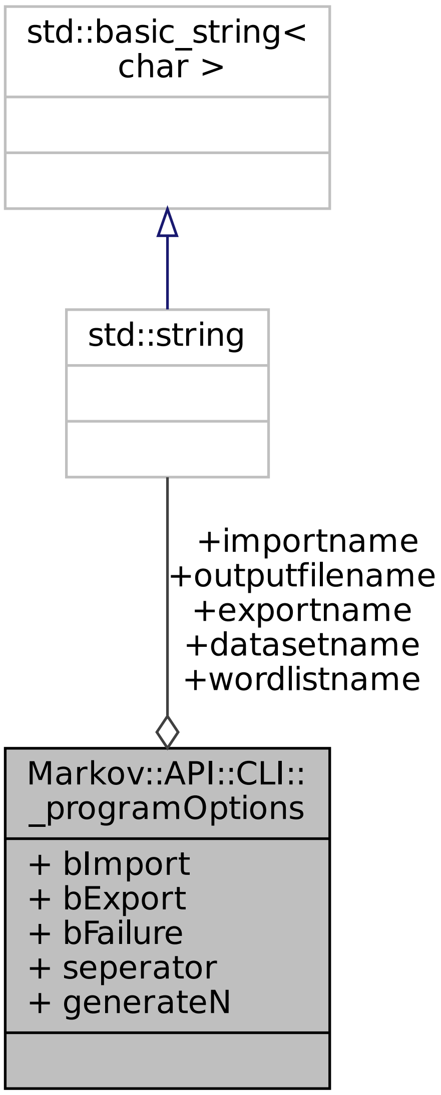
Class Documentation

8.1 Markov::API::CLI::_programOptions Struct Reference

Structure to hold parsed cli arguements.

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

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



Public Attributes

- bool bImport
- bool bExport
- bool bFailure
- char seperator
- std::string importname
- std::string exportname
- std::string wordlistname
- std::string outputfilename
- std::string datasetname
- int generateN

8.1.1 Detailed Description

Structure to hold parsed cli arguements.
Definition at line 18 of file argparse.h.

8.1.2 Member Data Documentation

8.1.2.1 bExport

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

8.1.2.2 bFailure

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

8.1.2.3 bImport

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

8.1.2.4 datasetname

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

8.1.2.5 exportname

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

8.1.2.6 generateN

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

8.1.2.7 importname

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

8.1.2.8 outputfilename

`std::string Markov::API::CLI::_programOptions::outputfilename`
Definition at line 26 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::Argparse(), and Markov::API::CLI::Argparse::setProgramOptions().

8.1.2.9 seperator

`char Markov::API::CLI::_programOptions::seperator`
Definition at line 22 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::setProgramOptions().

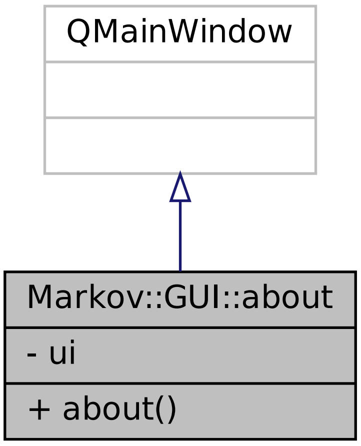
8.1.2.10 wordlistname

`std::string Markov::API::CLI::_programOptions::wordlistname`
Definition at line 25 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::Argparse().
The documentation for this struct was generated from the following file:

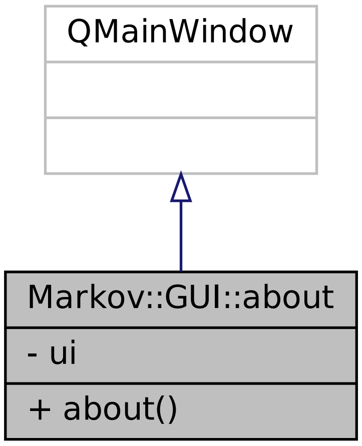
- argparse.h

8.2 Markov::GUI::about Class Reference

QT Class for about page.
`#include <about.h>`
Inheritance diagram for Markov::GUI::about:



Collaboration diagram for Markov::GUI::about:



Public Member Functions

- about (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::main ui

8.2.1 Detailed Description

QT Class for about page.
Definition at line 12 of file about.h.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 about()

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

8.2.3 Member Data Documentation

8.2.3.1 ui

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

Definition at line 18 of file about.h.
The documentation for this class was generated from the following file:

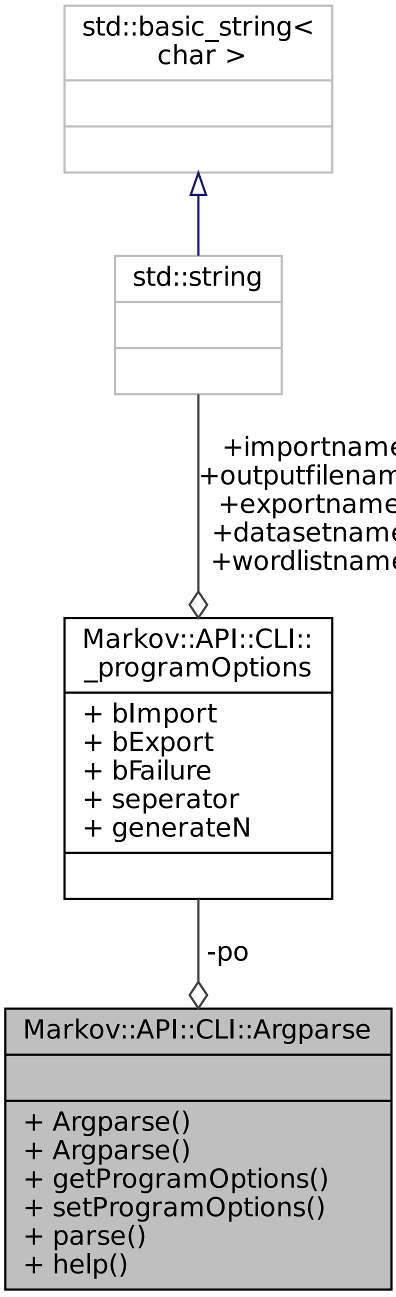
- about.h

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

Parse command line arguements.

`#include <argparse.h>`

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



Public Member Functions

- Argparse ()
- Argparse (int argc, char ∗∗argv)

Parse command line arguements.

- Markov::API::CLI::ProgramOptions getProgramOptions (void)

  Getter for command line options.

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

  Initialize program options structure.

Static Public Member Functions

- static Markov::API::CLI::ProgramOptions ∗ parse (int argc, char ∗∗argv)

  parse cli commands and return

- static void help ()

  Print help string.

Private Attributes

- Markov::API::CLI::ProgramOptions po

8.3.1 Detailed Description

Parse command line arguements.
Definition at line 34 of file argparse.h.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 Argparse() [1/2]

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

8.3.2.2 Argparse() [2/2]

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

Parse command line arguements.
Parses command line arguements to populate ProgramOptions structure.

Parameters

argc	Number of command line arguements
argv	Array of command line parameters

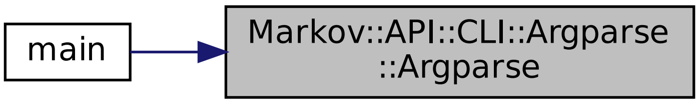
Definition at line 46 of file argparse.h.

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

```
            desc.add_options()
                ("generate", "Generate strings with given parameters")
                ("train", "Train model with given parameters")
                ("combine", "Combine")
                ("import", opt::value<std::string>(), "Import model file")
                ("output", opt::value<std::string>(), "Output model file. This model will be exported
when done. Will be ignored for generation mode")
                ("dataset", opt::value<std::string>(), "Dataset file to read input from training. Will
be ignored for generation mode")
                ("seperator", opt::value<char>(), "Seperator character to use with training data.
(character between occurence and value)")
                ("wordlist", opt::value<std::string>(), "Wordlist file path to export generation
results to. Will be ignored for training mode")
                ("count", opt::value<int>(), "Number of lines to generate. Ignored in training mode")
                ("verbosity", "Output verbosity")
                ("help", "Option definitions");

                opt::variables_map vm;

                opt::store(opt::parse_command_line(argc, argv, desc), vm);

                opt::notify(vm);

                //std::cout « desc « std::endl;
                if (vm.count("help")) {
                std::cout « desc « std::endl;
                }

                if (vm.count("output") == 0) this->po.outputfilename = "NULL";
                else if (vm.count("output") == 1) {
                    this->po.outputfilename = vm["output"].as<std::string>();
                    this->po.bExport = true;
                }
                else {
                    this->po.bFailure = true;
                    std::cout « "UNIDENTIFIED INPUT" « std::endl;
                    std::cout « desc « std::endl;
                }


                if (vm.count("dataset") == 0) this->po.datasetname = "NULL";
                else if (vm.count("dataset") == 1) {
                    this->po.datasetname = vm["dataset"].as<std::string>();
                }
                else {
                    this->po.bFailure = true;
                    std::cout « "UNIDENTIFIED INPUT" « std::endl;
                    std::cout « desc « std::endl;
                }


                if (vm.count("wordlist") == 0) this->po.wordlistname = "NULL";
                else if (vm.count("wordlist") == 1) {
                    this->po.wordlistname = vm["wordlist"].as<std::string>();
                }
                else {
                    this->po.bFailure = true;
                    std::cout « "UNIDENTIFIED INPUT" « std::endl;
                    std::cout « desc « std::endl;
                }

                if (vm.count("import") == 0) this->po.importname = "NULL";
                else if (vm.count("import") == 1) {
                    this->po.importname = vm["import"].as<std::string>();
                    this->po.bImport = true;
                }
                else {
                    this->po.bFailure = true;
                    std::cout « "UNIDENTIFIED INPUT" « std::endl;
                    std::cout « desc « std::endl;
                }


                if (vm.count("count") == 0) this->po.generateN = 0;
                else if (vm.count("count") == 1) {
                    this->po.generateN = vm["count"].as<int>();
                }
                else {
                    this->po.bFailure = true;
                    std::cout « "UNIDENTIFIED INPUT" « std::endl;
                    std::cout « desc « std::endl;
                }

                /*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::_program
Markov::API::CLI::_programOptions::datasetname, Markov::API::CLI::_programOptions::generateN, Markov::API::CLI::_programOptio
Markov::API::CLI::_programOptions::outputfilename, po, and Markov::API::CLI::_programOptions::wordlistname.
Referenced by main().
Here is the caller graph for this function:



8.3.3 Member Function Documentation

8.3.3.1 getProgramOptions()

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

Getter for command line options.
Getter for ProgramOptions populated by the arguement parser

Returns

ProgramOptions structure.

Definition at line 155 of file argparse.h.

```
00155                                                            {
00156             return this->po;
00157         }
```

References po.

8.3.3.2 help()

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

Print help string.
Definition at line 8 of file argparse.cpp.

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

```
    "   markov.exe -if empty_model.mdl -n 15000 -of wordlist.txt\n"
    "       import empty_model.mdl and generate 15000 words to wordlist.txt\n"

     « std::endl;
}
```

8.3.3.3 parse()

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

parse cli commands and return

Parameters

argc	- Program arguement count
argv	- Program arguement values array

Returns

ProgramOptions structure.

Definition at line 4 of file argparse.cpp.

```
{ return 0; }
```

8.3.3.4 setProgramOptions()

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

Initialize program options structure.

Parameters

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

Definition at line 172 of file argparse.h.

```

                                         {
            this->po.bImport = i;
            this->po.bExport = e;
            this->po.seperator = s;
            this->po.bFailure = bf;
            this->po.generateN = n;
            this->po.importname = iName;
            this->po.exportname = exName;
            this->po.outputfilename = oName;
            this->po.datasetname = dName;

            /*strcpy_s(this->po.importname,256,iName);
            strcpy_s(this->po.exportname,256,exName);
            strcpy_s(this->po.outputfilename,256,oName);
            strcpy_s(this->po.datasetname,256,dName);*/

        }
```

References Markov::API::CLI::_programOptions::bExport, Markov::API::CLI::_programOptions::bFailure, Markov::API::CLI::_program
Markov::API::CLI::_programOptions::datasetname, Markov::API::CLI::_programOptions::exportname, Markov::API::CLI::_programOp
Markov::API::CLI::_programOptions::importname, Markov::API::CLI::_programOptions::outputfilename, po, and Markov::API::CLI::_programOptions::seperator.

8.3.4 Member Data Documentation

8.3.4.1 po

`Markov::API::CLI::ProgramOptions Markov::API::CLI::Argparse::po [private]`

Definition at line 203 of file argparse.h.

Referenced by Argparse(), getProgramOptions(), and setProgramOptions().

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

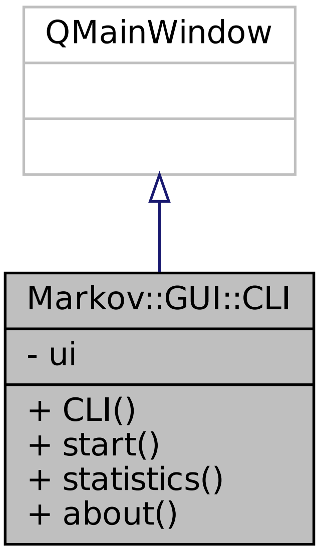
- argparse.h
- argparse.cpp

8.4 Markov::GUI::CLI Class Reference

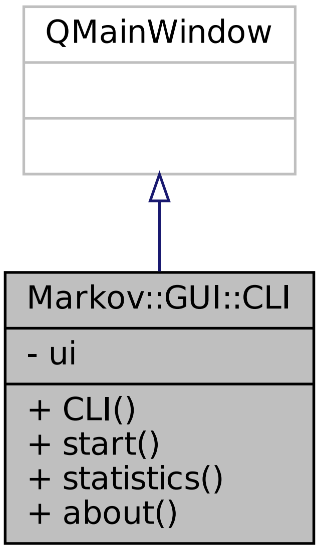
QT CLI Class.

`#include <CLI.h>`

Inheritance diagram for Markov::GUI::CLI:



Collaboration diagram for Markov::GUI::CLI:



Public Slots

- void start ()
- void statistics ()
- void about ()

Public Member Functions

- CLI (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::CLI ui

8.4.1 Detailed Description

QT CLI Class.
Definition at line 8 of file CLI.h.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 CLI()

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

8.4.3 Member Function Documentation

8.4.3.1 about

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

8.4.3.2 start

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

Referenced by main().

Here is the caller graph for this function:

main
Markov::GUI::CLI::start

8.4.3.3 statistics

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

8.4.4 Member Data Documentation

8.4.4.1 ui

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

Definition at line 14 of file CLI.h.

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

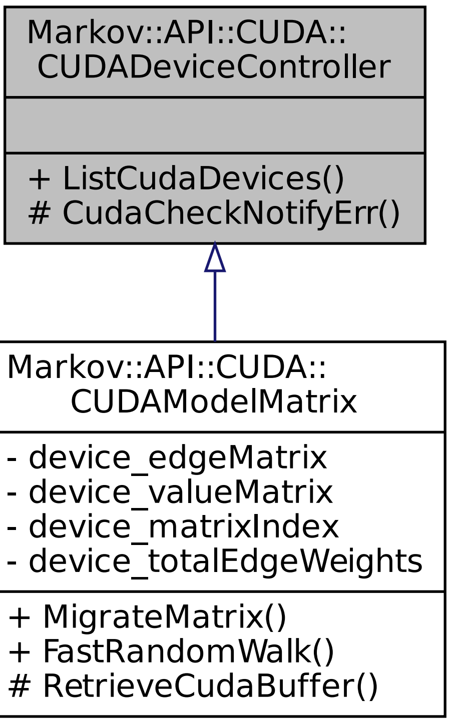
- CLI.h

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

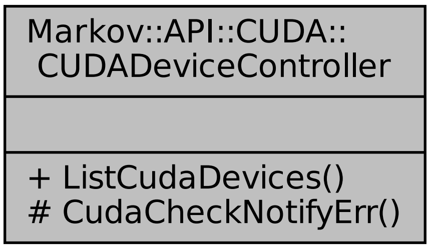
Controller class for CUDA device.

`#include <cudaDeviceController.h>`

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



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



Public Member Functions

- __host__ void ListCudaDevices ()

  List CUDA devices in the system.

Protected Member Functions

- __host__ int CudaCheckNotifyErr (cudaError_t _status, const char *msg)

  Check results of the last operation on GPU.

8.5.1 Detailed Description

Controller class for CUDA device.
This implementation only supports Nvidia devices.
Definition at line 9 of file cudaDeviceController.h.

8.5.2 Member Function Documentation

8.5.2.1 CudaCheckNotifyErr()

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

Check results of the last operation on GPU.
Check the status returned from cudaMalloc/cudaMemcpy to find failures.
If a failure occurs, its assumed beyond redemption, and exited.

Parameters

_status	Cuda error status to check
msg	Message to print in case of a failure

Returns

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

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

8.5.2.2 ListCudaDevices()

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

List CUDA devices in the system.
This function will print details of every CUDA capable device in the system.
Example output:

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

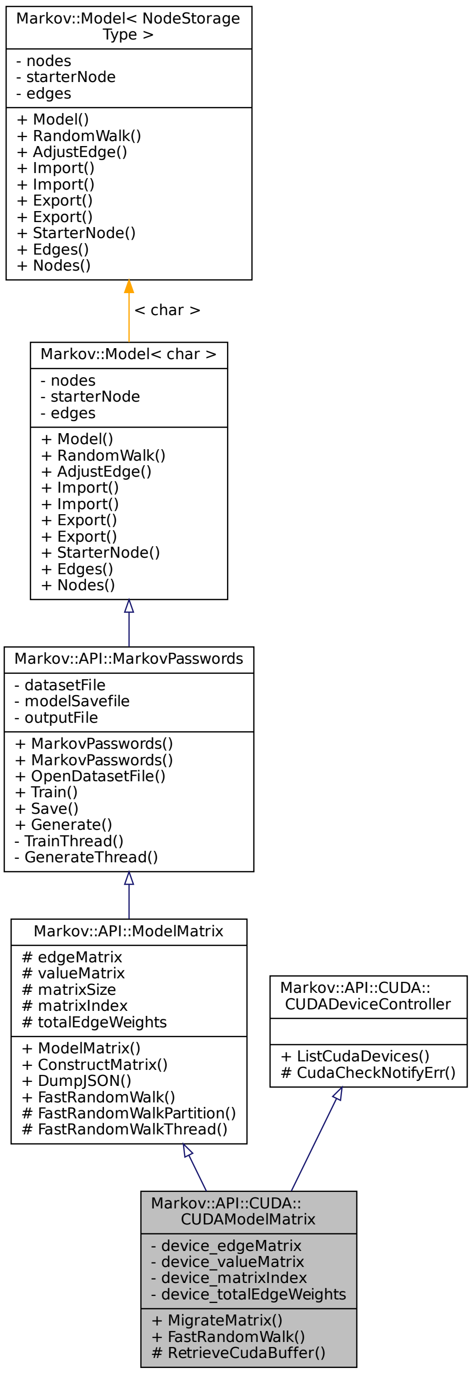
The documentation for this class was generated from the following file:

- cudaDeviceController.h

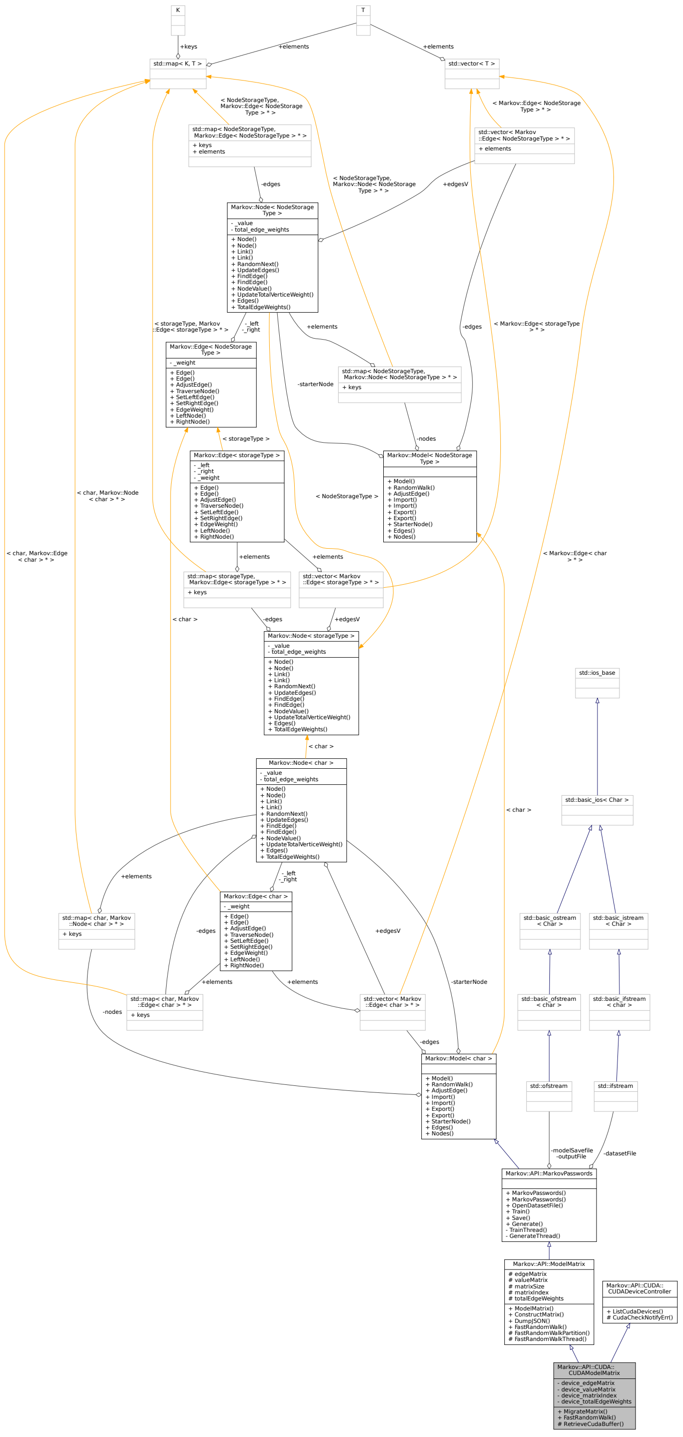
8.6 Markov::API::CUDA::CUDAModelMatrix Class Reference

Extension of Markov::API::ModelMatrix which is modified to run on GPU devices.
`#include <cudaModelMatrix.h>`

Inheritance diagram for Markov::API::CUDA::CUDAModelMatrix:



Collaboration diagram for Markov::API::CUDA::CUDAModelMatrix:



Public Member Functions

- __host__ void MigrateMatrix ()

    Migrate the class members to the VRAM.
- __host__ void FastRandomWalk (unsigned long int n, const char ∗wordlistFileName, int minLen, int maxLen, int threads, bool bFileIO)

    Random walk on the Matrix-reduced Markov::Model.
- void ConstructMatrix ()

Construct the related Matrix data for the model.
- void DumpJSON ()

  Debug function to dump the model to a JSON file.
- std::ifstream ∗ OpenDatasetFile (const char ∗filename)

  Open dataset file and return the ifstream pointer.
- void Train (const char ∗datasetFileName, char delimiter, int threads)

  Train the model with the dataset file.
- std::ofstream ∗ Save (const char ∗filename)

  Export model to file.
- void Generate (unsigned long int n, const char ∗wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

  Call Markov::Model::RandomWalk n times, and collect output.
- char ∗ RandomWalk (Markov::Random::RandomEngine ∗randomEngine, int minSetting, int maxSetting, char ∗buffer)

  Do a random walk on this model.
- void AdjustEdge (const char ∗payload, long int occurrence)

  Adjust the model with a single string.
- bool Import (std::ifstream ∗)

  Import a file to construct the model.
- bool Import (const char ∗filename)

  Open a file to import with filename, and call bool Model::Import with std::ifstream.
- bool Export (std::ofstream ∗)

  Export a file of the model.
- bool Export (const char ∗filename)

  Open a file to export with filename, and call bool Model::Export with std::ofstream.
- Node< char > ∗ StarterNode ()

  Return starter Node.
- std::vector< Edge< char > ∗ > ∗ Edges ()

  Return a vector of all the edges in the model.
- std::map< char, Node< char > ∗ > ∗ Nodes ()

  Return starter Node.
- __host__ void ListCudaDevices ()

  List CUDA devices in the system.

Protected Member Functions

- __host__ void RetrieveCudaBuffer ()

  Retrieve the result buffer from CUDA kernel.
- void FastRandomWalkPartition (std::mutex ∗mlock, std::ofstream ∗wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

  A single partition of FastRandomWalk event.
- void FastRandomWalkThread (std::mutex ∗mlock, std::ofstream ∗wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)

  A single thread of a single partition of FastRandomWalk.
- __host__ int CudaCheckNotifyErr (cudaError_t _status, const char ∗msg)

  Check results of the last operation on GPU.

Protected Attributes

- char ∗∗ edgeMatrix
- long int ∗∗ valueMatrix
- int matrixSize
- char ∗ matrixIndex
- long int ∗ totalEdgeWeights

Private Member Functions

- void TrainThread (Markov::API::Concurrency::ThreadSharedListHandler ∗listhandler, char delimiter)

  A single thread invoked by the Train function.
- void GenerateThread (std::mutex ∗outputLock, unsigned long int n, std::ofstream ∗wordlist, int minLen, int maxLen)

  A single thread invoked by the Generate function.

Private Attributes

- char ∗∗ device_edgeMatrix
- long int ∗∗ device_valueMatrix
- char ∗ device_matrixIndex
- long int ∗ device_totalEdgeWeights
- std::ifstream ∗ datasetFile
- std::ofstream ∗ modelSavefile
- std::ofstream ∗ outputFile
- std::map< char, Node< char > ∗ > nodes

  Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > ∗ starterNode

  Starter Node of this model.
- std::vector< Edge< char > ∗ > edges

  A list of all edges in this model.

8.6.1 Detailed Description

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

8.6.2 Member Function Documentation

8.6.2.1 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
                const NodeStorageType ∗ payload,
                long int occurrence ) [inherited]
```

Adjust the model with a single string.
Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.
Then, update the edge EdgeWeight from current node, to the terminator node.
This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.
Example Use: Create an empty model and train it with string: "testdata"

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

Parameters

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

Definition at line 322 of file model.h.

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

8.6.2.2 ConstructMatrix()

`void Markov::API::ModelMatrix::ConstructMatrix ( )  [inherited]`

Construct the related Matrix data for the model.

This operation can be used after importing/training to allocate and populate the matrix content.

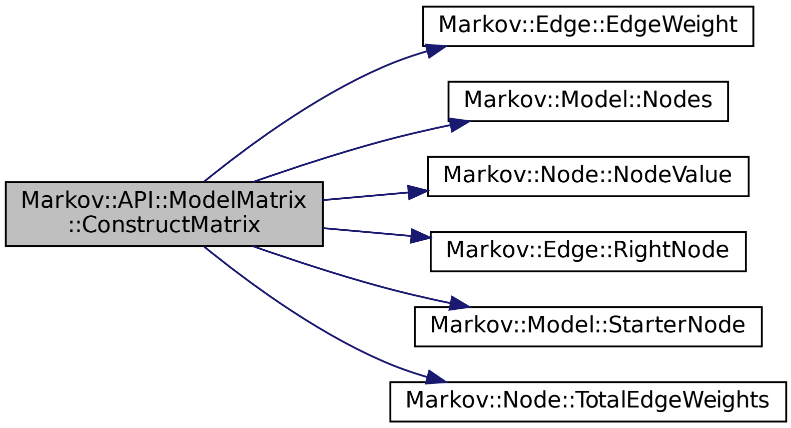
this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Definition at line 11 of file modelMatrix.cpp.

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

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

Markov::Node< storageType >::TotalEdgeWeights(), and Markov::API::ModelMatrix::valueMatrix.
Referenced by main().
Here is the call graph for this function:


Markov::API::ModelMatrix ::ConstructMatrix
Markov::Edge::EdgeWeight
Markov::Model::Nodes
Markov::Node::NodeValue
Markov::Edge::RightNode
Markov::Model::StarterNode
Markov::Node::TotalEdgeWeights

Here is the caller graph for this function:

main
Markov::API::ModelMatrix ::ConstructMatrix

8.6.2.3 CudaCheckNotifyErr()

```
__host__ int Markov::API::CUDA::CUDADeviceController::CudaCheckNotifyErr (
              cudaError_t _status,
              const char * msg )  [protected], [inherited]
```

Check results of the last operation on GPU.
Check the status returned from cudaMalloc/cudaMemcpy to find failures.
If a failure occurs, its assumed beyond redemption, and exited.

Parameters

_status	Cuda error status to check
msg	Message to print in case of a failure

Returns

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

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

8.6.2.4 DumpJSON()

`void Markov::API::ModelMatrix::DumpJSON ( )  [inherited]`

Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.

Definition at line 60 of file modelMatrix.cpp.

```
00060                                {
00061
00062     std::cout « "{\n   \"index\": \"";
00063     for(int i=0;i<this->matrixSize;i++){
00064         if(this->matrixIndex[i]==’"’) std::cout « "\\\"";
00065         else if(this->matrixIndex[i]==’\\’) std::cout « "\\\\";
00066         else if(this->matrixIndex[i]==0) std::cout « "\\\\x00";
00067         else if(i==0) std::cout « "\\\\xff";
00068         else if(this->matrixIndex[i]==’\n’) std::cout « "\\n";
00069         else std::cout « this->matrixIndex[i];
00070     }
00071     std::cout «
00072     "\",\n"
00073     "    \"edgemap\": {\n";
00074
00075     for(int i=0;i<this->matrixSize;i++){
00076         if(this->matrixIndex[i]==’"’) std::cout « "       \"\\\"\": [";
00077         else if(this->matrixIndex[i]==’\\’) std::cout « "       \"\\\\\": [";
00078         else if(this->matrixIndex[i]==0) std::cout « "       \"\\\\x00\": [";
00079         else if(this->matrixIndex[i]<0) std::cout « "       \"\\\\xff\": [";
00080         else std::cout « "       \"" « this->matrixIndex[i] « "\": [";
00081         for(int j=0;j<this->matrixSize;j++){
00082             if(this->edgeMatrix[i][j]==’"’) std::cout « "\"\\\"\"";
00083             else if(this->edgeMatrix[i][j]==’\\’) std::cout « "\"\\\\\"";
00084             else if(this->edgeMatrix[i][j]==0) std::cout « "\"\\\\x00\"";
00085             else if(this->edgeMatrix[i][j]<0) std::cout « "\"\\\\xff\"";
00086             else if(this->matrixIndex[i]==’\n’) std::cout « "\"\\n\"";
00087             else std::cout « "\"" « this->edgeMatrix[i][j] « "\"";
00088             if(j!=this->matrixSize-1) std::cout « ", ";
00089         }
00090         std::cout « "],\n";
00091     }
00092     std::cout « "},\n";
00093
00094     std::cout « "\"   weightmap\": {\n";
00095     for(int i=0;i<this->matrixSize;i++){
00096         if(this->matrixIndex[i]==’"’) std::cout « "       \"\\\"\": [";
00097         else if(this->matrixIndex[i]==’\\’) std::cout « "       \"\\\\\": [";
00098         else if(this->matrixIndex[i]==0) std::cout « "       \"\\\\x00\": [";
00099         else if(this->matrixIndex[i]<0) std::cout « "       \"\\\\xff\": [";
00100         else std::cout « "       \"" « this->matrixIndex[i] « "\": [";
00101
00102         for(int j=0;j<this->matrixSize;j++){
00103             std::cout « this->valueMatrix[i][j];
00104             if(j!=this->matrixSize-1) std::cout « ", ";
00105         }
00106         std::cout « "],\n";
00107     }
00108     std::cout « "  }\n}\n";
00109 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, and Markov::API::ModelMatrix::valueMatrix.

8.6.2.5 Edges()

`std::vector<Edge<char >*>* Markov::Model< char >::Edges ( )  [inline], [inherited]`

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 172 of file model.h.

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

8.6.2.6 Export() [1/2]

`bool Markov::Model< char >::Export (`

```
              const char * filename )  [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

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

Definition at line 285 of file model.h.

```
                                                              {
    std::ofstream exportfile;
    exportfile.open(filename);
    return this->Export(&exportfile);
}
```

8.6.2.7 Export() [2/2]

```
bool Markov::Model< char >::Export (
              std::ofstream * f )  [inherited]
```

Export a file of the model.
File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.
Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

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

Definition at line 273 of file model.h.

```
                                                        {
    Markov::Edge<NodeStorageType>* e;
    for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
        e = this->edges[i];
        //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
   e->RightNode()->NodeValue() « "\n";
        *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
   "\n";
    }

    return true;
}
```

8.6.2.8 FastRandomWalk()

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

Random walk on the Matrix-reduced Markov::Model.
TODO

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

Parameters

threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

8.6.2.9 FastRandomWalkPartition()

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

A single partition of FastRandomWalk event.
Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation
- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

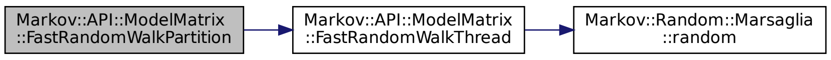
Parameters

mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

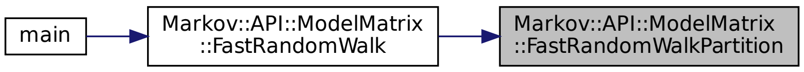
Definition at line 182 of file modelMatrix.cpp.

```
                                                                    {

    int iterationsPerThread = n/threads;
    int iterationsPerThreadCarryOver = n%threads;

    std::vector<std::thread*> threadsV;

    int id = 0;
    for(int i=0;i<threads;i++){
        threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
    mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
        id++;
    }

    threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
    wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));

    for(int i=0;i<threads;i++){
        threadsV[i]->join();
    }
}
```

References Markov::API::ModelMatrix::FastRandomWalkThread().
Referenced by Markov::API::ModelMatrix::FastRandomWalk().
Here is the call graph for this function:



Here is the caller graph for this function:



8.6.2.10 FastRandomWalkThread()

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

A single thread of a single partition of FastRandomWalk.
A FastRandomWalkPartition will initiate as many of this function as requested.
This function contains the bulk of the generation algorithm.

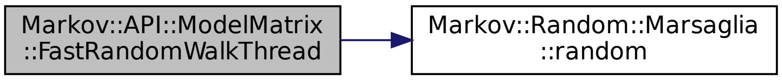
Parameters

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

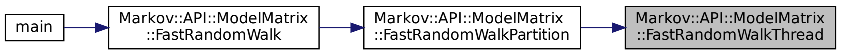
Definition at line 112 of file modelMatrix.cpp.

```
                                                                                                                    {
    if(n==0) return;

    Markov::Random::Marsaglia MarsagliaRandomEngine;
    char* e;
    char *res = new char[maxLen*n];
    int index = 0;
    char next;
    int len=0;
    long int selection;
    char cur;
    long int bufferctr = 0;
    for (int i = 0; i < n; i++) {
        cur=199;
        len=0;
        while (true) {
            e = strchr(this->matrixIndex, cur);
            index = e - this->matrixIndex;
            selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
            for(int j=0;j<this->matrixSize;j++){
                selection -= this->valueMatrix[index][j];
                if (selection < 0){
                    next = this->edgeMatrix[index][j];
                    break;
                }
            }

            if (len >= maxLen)  break;
            else if ((next < 0) && (len < minLen)) continue;
            else if (next < 0) break;
            cur = next;
            res[bufferctr + len++] = cur;
        }
        res[bufferctr + len++] = '\n';
        bufferctr+=len;

    }
    if(bFileIO){
        mlock->lock();
        *wordlist « res;
        mlock->unlock();
    }else{
        mlock->lock();
        std::cout « res;
        mlock->unlock();
    }
    delete res;

}
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, Markov::Random::Marsaglia::random(), Markov::API::ModelMatrix::totalEdgeWeights, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::API::ModelMatrix::FastRandomWalkPartition().
Here is the call graph for this function:


Markov::API::ModelMatrix ::FastRandomWalkThread
Markov::Random::Marsaglia ::random

Here is the caller graph for this function:


main
Markov::API::ModelMatrix ::FastRandomWalk
Markov::API::ModelMatrix ::FastRandomWalkPartition
Markov::API::ModelMatrix ::FastRandomWalkThread

8.6.2.11 Generate()

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

Call Markov::Model::RandomWalk n times, and collect output.
Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

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

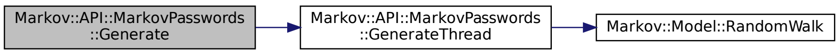
Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

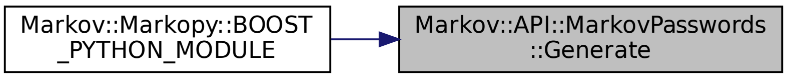
Definition at line 92 of file markovPasswords.cpp.

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

    for(int i=0;i<threads;i++){
        threadsV[i]->join();
        delete threadsV[i];
    }

    this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);

}
```

References Markov::API::MarkovPasswords::GenerateThread().
Referenced by Markov::Markopy::BOOST_PYTHON_MODULE().
Here is the call graph for this function:


Markov::API::MarkovPasswords ::Generate
Markov::API::MarkovPasswords ::GenerateThread
Markov::Model::RandomWalk

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::API::MarkovPasswords ::Generate

8.6.2.12 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
            std::mutex * outputLock,
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen,
            int maxLen )  [private], [inherited]
```

A single thread invoked by the Generate function.

DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Parameters

outputLock	- shared mutex lock to lock during output operation. Prevents race condition on write.
n	number of lines to be generated by this thread
wordlist	wordlistfile
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

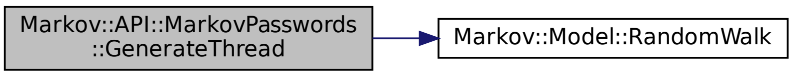
Definition at line 114 of file markovPasswords.cpp.

```
                                               {
    char* res = new char[maxLen+5];
    if(n==0) return;

    Markov::Random::Marsaglia MarsagliaRandomEngine;
    for (int i = 0; i < n; i++) {
        this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
        outputLock->lock();
        *wordlist « res « "\n";
        outputLock->unlock();
    }
}
```

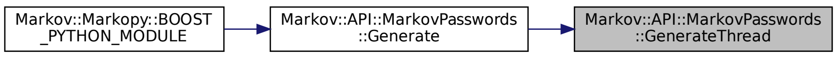
References Markov::Model< NodeStorageType >::RandomWalk().

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

Here is the call graph for this function:


Markov::API::MarkovPasswords ::GenerateThread
Markov::Model::RandomWalk

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::API::MarkovPasswords ::Generate
Markov::API::MarkovPasswords ::GenerateThread

8.6.2.13 Import() [1/2]

```
bool Markov::Model< char >::Import (
                const char * filename )  [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

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

Definition at line 265 of file model.h.

```
                                                              {
    std::ifstream importfile;
    importfile.open(filename);
    return this->Import(&importfile);

}
```

8.6.2.14 Import() [2/2]

```
bool Markov::Model< char >::Import (
              std::ifstream * f )  [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

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

Definition at line 206 of file model.h.

```
                                                        {
    std::string cell;

    char src;
    char target;
    long int oc;

    while (std::getline(*f, cell)) {
        //std::cout « "cell: " « cell « std::endl;
        src = cell[0];
        target = cell[cell.length() - 1];
        char* j;
        oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
        //std::cout « oc « "\n";
        Markov::Node<NodeStorageType>* srcN;
        Markov::Node<NodeStorageType>* targetN;
        Markov::Edge<NodeStorageType>* e;
        if (this->nodes.find(src) == this->nodes.end()) {
            srcN = new Markov::Node<NodeStorageType>(src);
            this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
            //std::cout « "Creating new node at start.\n";
        }
        else {
            srcN = this->nodes.find(src)->second;
        }

        if (this->nodes.find(target) == this->nodes.end()) {
            targetN = new Markov::Node<NodeStorageType>(target);
            this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
            //std::cout « "Creating new node at end.\n";
        }
        else {
            targetN = this->nodes.find(target)->second;
        }
        e = srcN->Link(targetN);
        e->AdjustEdge(oc);
        this->edges.push_back(e);

        //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
   int(targetN->NodeValue()) « "\n";


    }

    for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
        //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
        std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
    Edge<NodeStorageType> *rhs)->bool{
            return lhs->EdgeWeight() > rhs->EdgeWeight();
        });
        //for(int i=0;i<x.second->edgesV.size();i++)
        //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
        //std::cout « "\n";
    }
    //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
    //std::cout « "Total number of edges: " « this->edges.size() « std::endl;

    return true;
}
```

8.6.2.15 ListCudaDevices()

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

List CUDA devices in the system.

This function will print details of every CUDA capable device in the system.

Example output:

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

8.6.2.16 MigrateMatrix()

`__host__ void Markov::API::CUDA::CUDAModelMatrix::MigrateMatrix ( )`

Migrate the class members to the VRAM.

Cannot be used without calling Markov::API::ModelMatrix::ConstructMatrix at least once. This function will manage the memory allocation and data transfer from CPU RAM to GPU VRAM.

Newly allocated VRAM pointers are set in the class member variables.

8.6.2.17 Nodes()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 177 of file model.h.

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

8.6.2.18 OpenDatasetFile()

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

ifstream* to the the dataset file

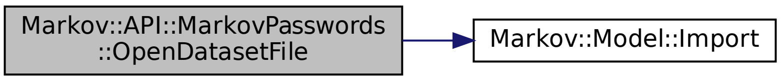
Definition at line 27 of file markovPasswords.cpp.

```
                                                                      {

    std::ifstream* datasetFile;
```

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

References Markov::Model< NodeStorageType >::Import().

Here is the call graph for this function:



8.6.2.19 RandomWalk()

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 292 of file model.h.

```
                                                                                  {
    Markov::Node<NodeStorageType>* n = this->starterNode;
    int len = 0;
    Markov::Node<NodeStorageType>* 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();
    }

    //null terminate the string
    buffer[len] = 0x00;

    //do something with the generated string
    return buffer; //for now
}
```

8.6.2.20 RetrieveCudaBuffer()

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

Retrieve the result buffer from CUDA kernel.

Done on each partition.

8.6.2.21 Save()

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

Export model to file.

Parameters

filename	- Export filename.

Returns

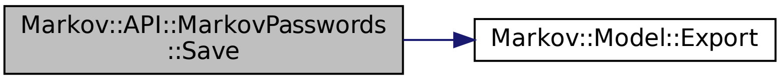
std::ofstream* of the exported file.

Definition at line 80 of file markovPasswords.cpp.

```
                                                                                  {
    std::ofstream* exportFile;

    std::ofstream newFile(filename);

    exportFile = &newFile;

    this->Export(exportFile);
    return exportFile;
}
```

References Markov::Model< NodeStorageType >::Export().

Here is the call graph for this function:



8.6.2.22 StarterNode()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 167 of file model.h.

```
{ return starterNode;}
```

8.6.2.23 Train()

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

Train the model with the dataset file.

Parameters

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

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

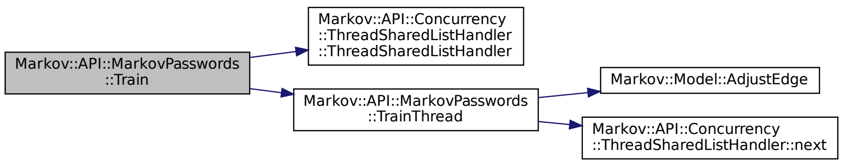
Definition at line 40 of file markovPasswords.cpp.

```
                                                                                    {
    Markov::API::Concurrency::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(&Markov::API::MarkovPasswords::TrainThread, this,
  &listhandler, 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";

}
```

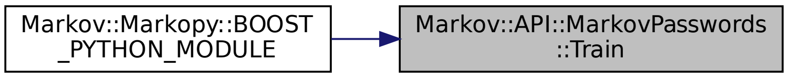
References Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler(), and Markov::API::MarkovPasswords::

Referenced by Markov::Markopy::BOOST_PYTHON_MODULE().

Here is the call graph for this function:


Markov::API::MarkovPasswords ::Train
Markov::API::Concurrency ::ThreadSharedListHandler ::ThreadSharedListHandler
Markov::API::MarkovPasswords ::TrainThread
Markov::Model::AdjustEdge
Markov::API::Concurrency ::ThreadSharedListHandler::next

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::API::MarkovPasswords ::Train

8.6.2.24 TrainThread()

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

A single thread invoked by the Train function.

Parameters

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

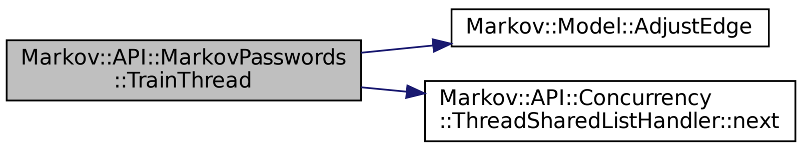
Definition at line 59 of file markovPasswords.cpp.

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

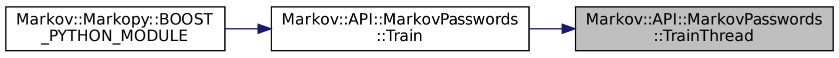
References Markov::Model< NodeStorageType >::AdjustEdge(), and Markov::API::Concurrency::ThreadSharedListHandler::next().

Referenced by Markov::API::MarkovPasswords::Train().

Here is the call graph for this function:


Markov::API::MarkovPasswords ::TrainThread
Markov::Model::AdjustEdge
Markov::API::Concurrency ::ThreadSharedListHandler::next

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::API::MarkovPasswords ::Train
Markov::API::MarkovPasswords ::TrainThread

8.6.3 Member Data Documentation

8.6.3.1 datasetFile

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

8.6.3.2 device_edgeMatrix

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

8.6.3.3 device_matrixIndex

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

8.6.3.4 device_totalEdgeWeights

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

8.6.3.5 device_valueMatrix

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

8.6.3.6 edgeMatrix

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

Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.6.3.7 edges

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

A list of all edges in this model.

Definition at line 194 of file model.h.

8.6.3.8 matrixIndex

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

Definition at line 115 of file modelMatrix.h.

Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.6.3.9 matrixSize

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

Definition at line 114 of file modelMatrix.h.

Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.6.3.10 modelSavefile

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

Definition at line 107 of file markovPasswords.h.

8.6.3.11 nodes

`std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]`

Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.

Definition at line 183 of file model.h.

8.6.3.12 outputFile

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

Definition at line 108 of file markovPasswords.h.

8.6.3.13 starterNode

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

Starter Node of this model.

Definition at line 188 of file model.h.

8.6.3.14 totalEdgeWeights

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

Definition at line 116 of file modelMatrix.h.

Referenced by Markov::API::ModelMatrix::ConstructMatrix(), and Markov::API::ModelMatrix::FastRandomWalkThread().

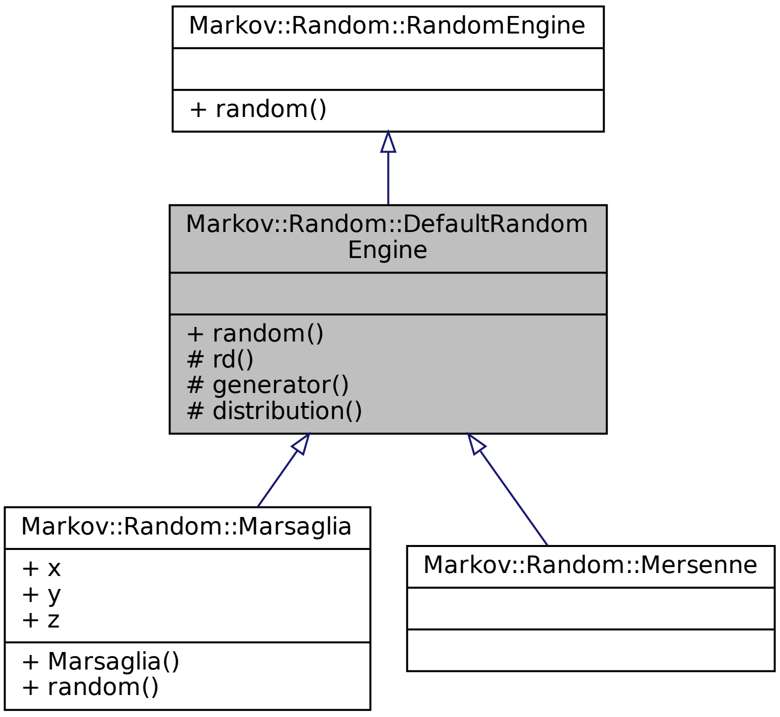
8.6.3.15 valueMatrix

`long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]`
Definition at line 113 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().
The documentation for this class was generated from the following file:

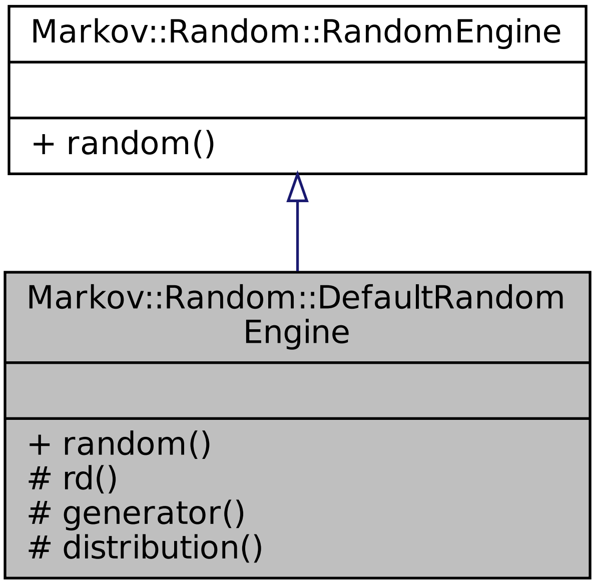
- cudaModelMatrix.h

8.7 Markov::Random::DefaultRandomEngine Class Reference

Implementation using Random.h default random engine.
`#include <random.h>`
Inheritance diagram for Markov::Random::DefaultRandomEngine:



Collaboration diagram for Markov::Random::DefaultRandomEngine:


Markov::Random::RandomEngine
+ random()
Markov::Random::DefaultRandom Engine
+ random()
# rd()
# generator()
# distribution()

Public Member Functions

- unsigned long random ()

  Generate Random Number.

Protected Member Functions

- std::random_device & rd ()

  Default random device for seeding.
- std::default_random_engine & generator ()

  Default random engine for seeding.
- std::uniform_int_distribution< long long unsigned > & distribution ()

  Distribution schema for seeding.

8.7.1 Detailed Description

Implementation using Random.h default random engine.
This engine is also used by other engines for seeding.
Example Use: Using Default Engine with RandomWalk
```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::DefaultRandomEngine randomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&randomEngine, 5, 10, res);
    std::cout « res « "\n";
 }
```
Example Use: Generating a random number with Marsaglia Engine
```
Markov::Random::DefaultRandomEngine de;
std::cout « de.random();
```
Definition at line 52 of file random.h.

8.7.2 Member Function Documentation

8.7.2.1 distribution()

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

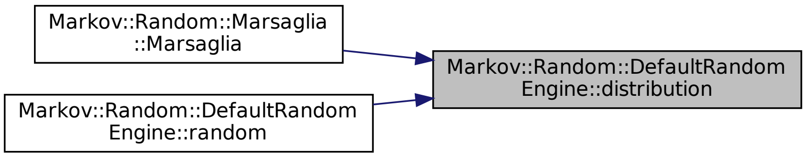
Distribution schema for seeding.

Definition at line 81 of file random.h.

```
                                                                                {
            static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffFFFF);
            return _distribution;
        }
```

Referenced by Markov::Random::Marsaglia::Marsaglia(), and random().

Here is the caller graph for this function:


Markov::Random::Marsaglia ::Marsaglia
Markov::Random::DefaultRandom Engine::random
Markov::Random::DefaultRandom Engine::distribution

8.7.2.2 generator()

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

Default random engine for seeding.

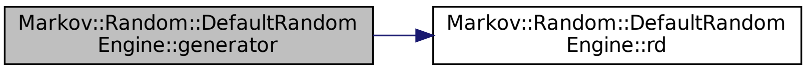
Definition at line 73 of file random.h.

```
                                                    {
            static std::default_random_engine _generator(rd()());
            return _generator;
        }
```

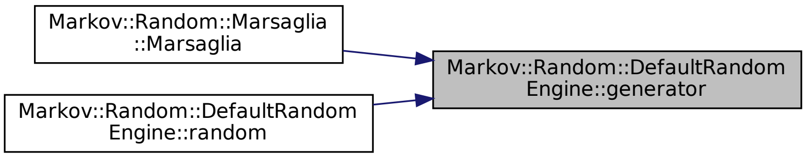
References rd().

Referenced by Markov::Random::Marsaglia::Marsaglia(), and random().

Here is the call graph for this function:


Markov::Random::DefaultRandom Engine::generator
Markov::Random::DefaultRandom Engine::rd

Here is the caller graph for this function:


Markov::Random::Marsaglia ::Marsaglia
Markov::Random::DefaultRandom Engine::random
Markov::Random::DefaultRandom Engine::generator

8.7.2.3 random()

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

Generate Random Number.

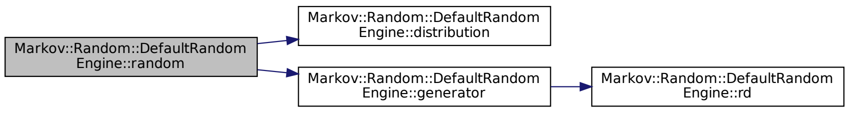
Returns

random number in long range.

Implements Markov::Random::RandomEngine.
Reimplemented in Markov::Random::Marsaglia.
Definition at line 57 of file random.h.

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

References distribution(), and generator().
Here is the call graph for this function:


Markov::Random::DefaultRandom Engine::random
Markov::Random::DefaultRandom Engine::distribution
Markov::Random::DefaultRandom Engine::generator
Markov::Random::DefaultRandom Engine::rd

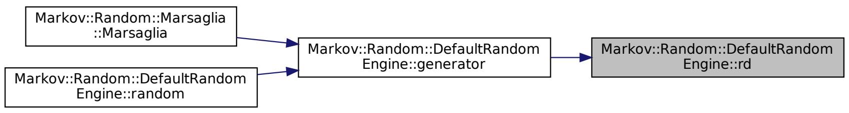
8.7.2.4 rd()

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

Default random device for seeding.
Definition at line 65 of file random.h.

```
                                      {
            static std::random_device _rd;
            return _rd;
        }
```

Referenced by generator().
Here is the caller graph for this function:


Markov::Random::Marsaglia ::Marsaglia
Markov::Random::DefaultRandom Engine::random
Markov::Random::DefaultRandom Engine::generator
Markov::Random::DefaultRandom Engine::rd

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

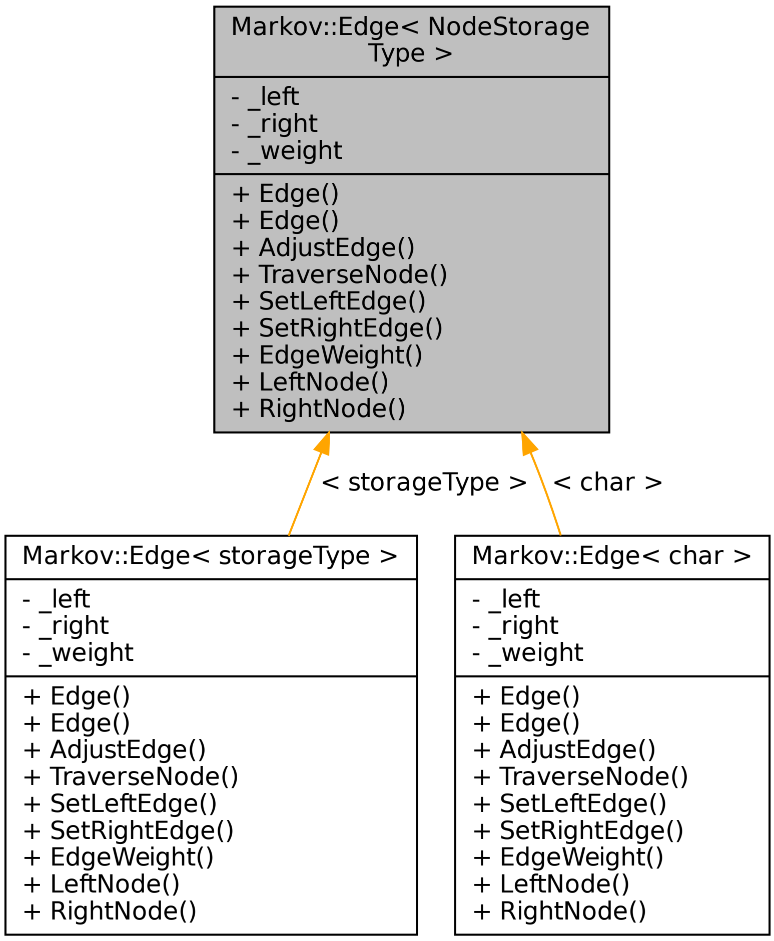
- random.h

8.8 Markov::Edge< NodeStorageType > Class Template Reference

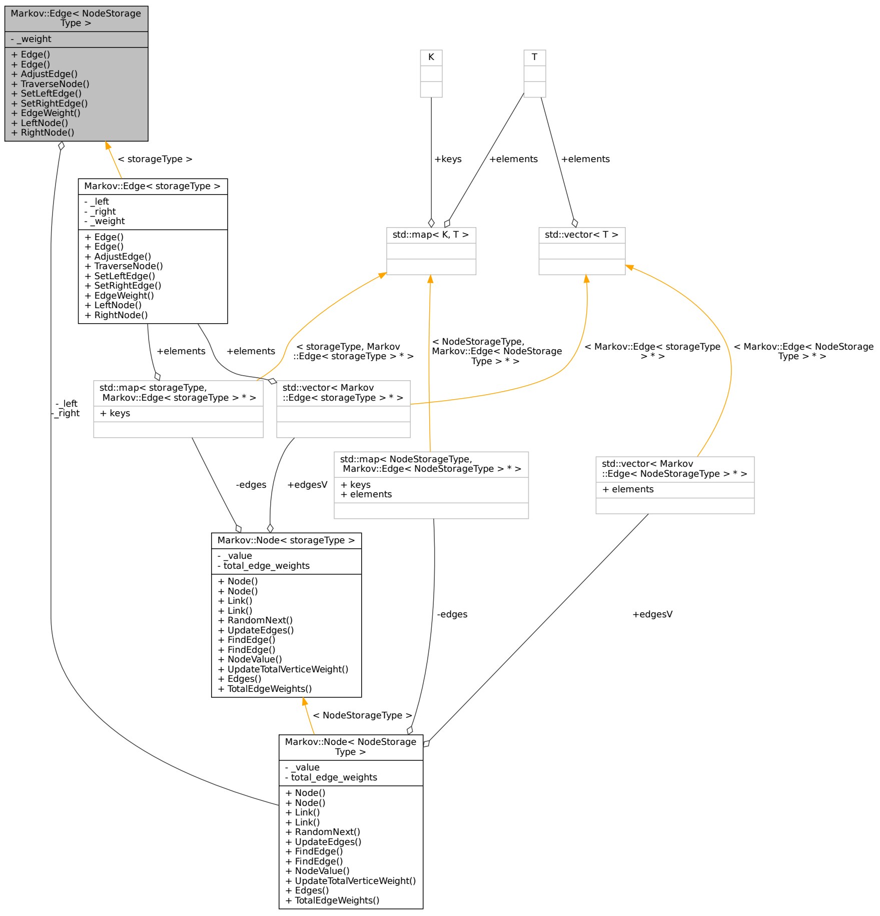
Edge class used to link nodes in the model together.

`#include <model.h>`

Inheritance diagram for Markov::Edge< NodeStorageType >:



Collaboration diagram for Markov::Edge< NodeStorageType >:


Markov::Edge< NodeStorage Type >
- _weight
+ Edge()
+ Edge()
+ AdjustEdge()
+ TraverseNode()
+ SetLeftEdge()
+ SetRightEdge()
+ EdgeWeight()
+ LeftNode()
+ RightNode()
< storageType >
Markov::Edge< storageType >
- _left
- _right
- _weight
K
T
+keys
+elements
+elements
std::map< K, T >
std::vector< T >
+elements
+elements
< storageType, Markov ::Edge< storageType > * >
< NodeStorageType, Markov::Edge< NodeStorage Type > * >
< Markov::Edge< storageType > * >
< Markov::Edge< NodeStorage Type > * >
- _left
- _right
std::map< storageType, Markov::Edge< storageType > * >
+ keys
std::vector< Markov ::Edge< storageType > * >
std::map< NodeStorageType, Markov::Edge< NodeStorageType > * >
+ keys
+ elements
std::vector< Markov ::Edge< NodeStorageType > * >
+ elements
-edges
+edgesV
Markov::Node< storageType >
- _value
- total_edge_weights
+ Node()
+ Node()
+ Link()
+ Link()
+ RandomNext()
+ UpdateEdges()
+ FindEdge()
+ FindEdge()
+ NodeValue()
+ UpdateTotalVerticeWeight()
+ Edges()
+ TotalEdgeWeights()
-edges
+edgesV
< NodeStorageType >
Markov::Node< NodeStorage Type >

Public Member Functions

- Edge ()

  Default constructor.
- Edge (Node< NodeStorageType > ∗_left, Node< NodeStorageType > ∗_right)

  Constructor. Initialize edge with given RightNode and LeftNode.
- void AdjustEdge (long int offset)

  Adjust the edge EdgeWeight with offset. Adds the offset parameter to the edge EdgeWeight.
- Node< NodeStorageType > ∗ TraverseNode ()

  Traverse this edge to RightNode.
- void SetLeftEdge (Node< NodeStorageType > ∗)

  Set LeftNode of this edge.
- void SetRightEdge (Node< NodeStorageType > ∗)

  Set RightNode of this edge.
- uint64_t EdgeWeight ()

  return edge's EdgeWeight.
- Node< NodeStorageType > ∗ LeftNode ()

  return edge's LeftNode
- Node< NodeStorageType > ∗ RightNode ()

  return edge's RightNode

Private Attributes

- Node< NodeStorageType > ∗ _left
- Node< NodeStorageType > ∗ _right

  source node
- long int _weight

  target node

8.8.1 Detailed Description

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

Edge class used to link nodes in the model together.
Has LeftNode, RightNode, and EdgeWeight of the edge. Edges are *UNIDIRECTIONAL* in this model. They can only be traversed LeftNode to RightNode.
Definition at line 26 of file model.h.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 Edge() [1/2]

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

Default constructor.

Definition at line 105 of file edge.h.

```
                                       {
    this->_left = NULL;
    this->_right = NULL;
    this->_weight = 0;
}
```

8.8.2.2 Edge() [2/2]

```
template<typename NodeStorageType >
Markov::Edge< NodeStorageType >::Edge (
              Markov::Node< NodeStorageType > * _left,
              Markov::Node< NodeStorageType > * _right )
```

Constructor. Initialize edge with given RightNode and LeftNode.

Parameters

_left	- Left node of this edge.
_right	- Right node of this edge.

Example Use: Construct edge

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

Definition at line 112 of file edge.h.

```

         {
    this->_left = _left;
    this->_right = _right;
    this->_weight = 0;
}
```

8.8.3 Member Function Documentation

8.8.3.1 AdjustEdge()

```
template<typename NodeStorageType >
void Markov::Edge< NodeStorageType >::AdjustEdge (
              long int offset )
```

Adjust the edge EdgeWeight with offset. Adds the offset parameter to the edge EdgeWeight.

Parameters

offset	- 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.

```
                                                                {
    this->_weight += offset;
    this->LeftNode()->UpdateTotalVerticeWeight(offset);
}
```

8.8.3.2 EdgeWeight()

```
template<typename NodeStorageType >
uint64_t Markov::Edge< NodeStorageType >::EdgeWeight  [inline]
```

return edge's EdgeWeight.

Returns

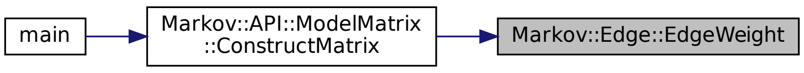
edge's EdgeWeight.

Definition at line 142 of file edge.h.

```
                                                            {
    return this->_weight;
}
```

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

Here is the caller graph for this function:



8.8.3.3 LeftNode()

```
template<typename NodeStorageType >
Markov::Node< NodeStorageType > * Markov::Edge< NodeStorageType >::LeftNode
```

return edge's LeftNode

Returns

edge's LeftNode.

Definition at line 147 of file edge.h.

```
                                                                          {
    return this->_left;
}
```

8.8.3.4 RightNode()

```
template<typename NodeStorageType >
Markov::Node< NodeStorageType > * Markov::Edge< NodeStorageType >::RightNode  [inline]
```

return edge's RightNode

Returns

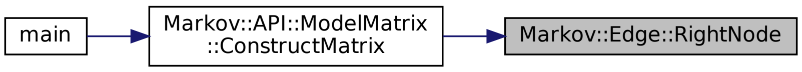
edge's RightNode.

Definition at line 152 of file edge.h.

```
                                                                                {
    return this->_right;
}
```

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

Here is the caller graph for this function:



8.8.3.5 SetLeftEdge()

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

Set LeftNode of this edge.

Parameters

node	- Node to be linked with.

Definition at line 132 of file edge.h.

```
                                                                                 {
    this->_left = n;
}
```

8.8.3.6 SetRightEdge()

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

Set RightNode of this edge.

Parameters

node	- Node to be linked with.

Definition at line 137 of file edge.h.

```
                                                                                  {
    this->_right = n;
}
```

8.8.3.7 TraverseNode()

```
template<typename NodeStorageType >
Markov::Node< NodeStorageType > * Markov::Edge< NodeStorageType >::TraverseNode  [inline]
```

Traverse this edge to RightNode.

Returns

Right node. If this is a terminator node, return NULL

Example Use: Traverse a node

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

Definition at line 125 of file edge.h.

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

8.8.4 Member Data Documentation

8.8.4.1 _left

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

Definition at line 95 of file edge.h.

8.8.4.2 _right

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

source node

Definition at line 96 of file edge.h.

Referenced by Markov::Edge< char >::TraverseNode().

8.8.4.3 _weight

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

target node

Definition at line 97 of file edge.h.

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

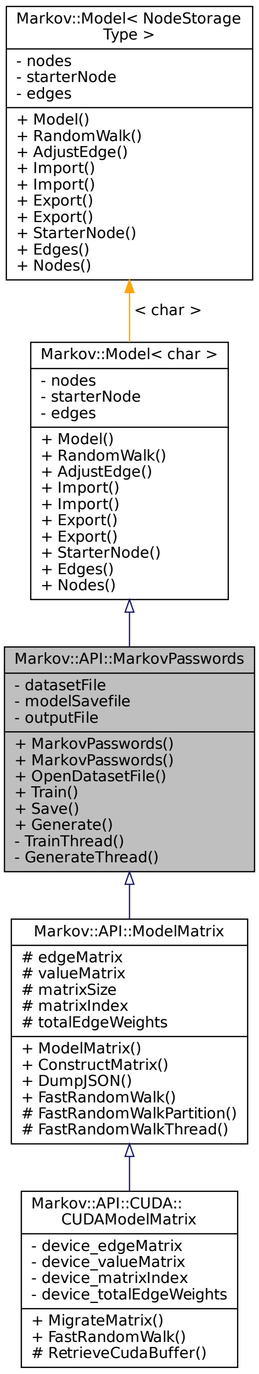
- model.h
- edge.h

8.9 Markov::API::MarkovPasswords Class Reference

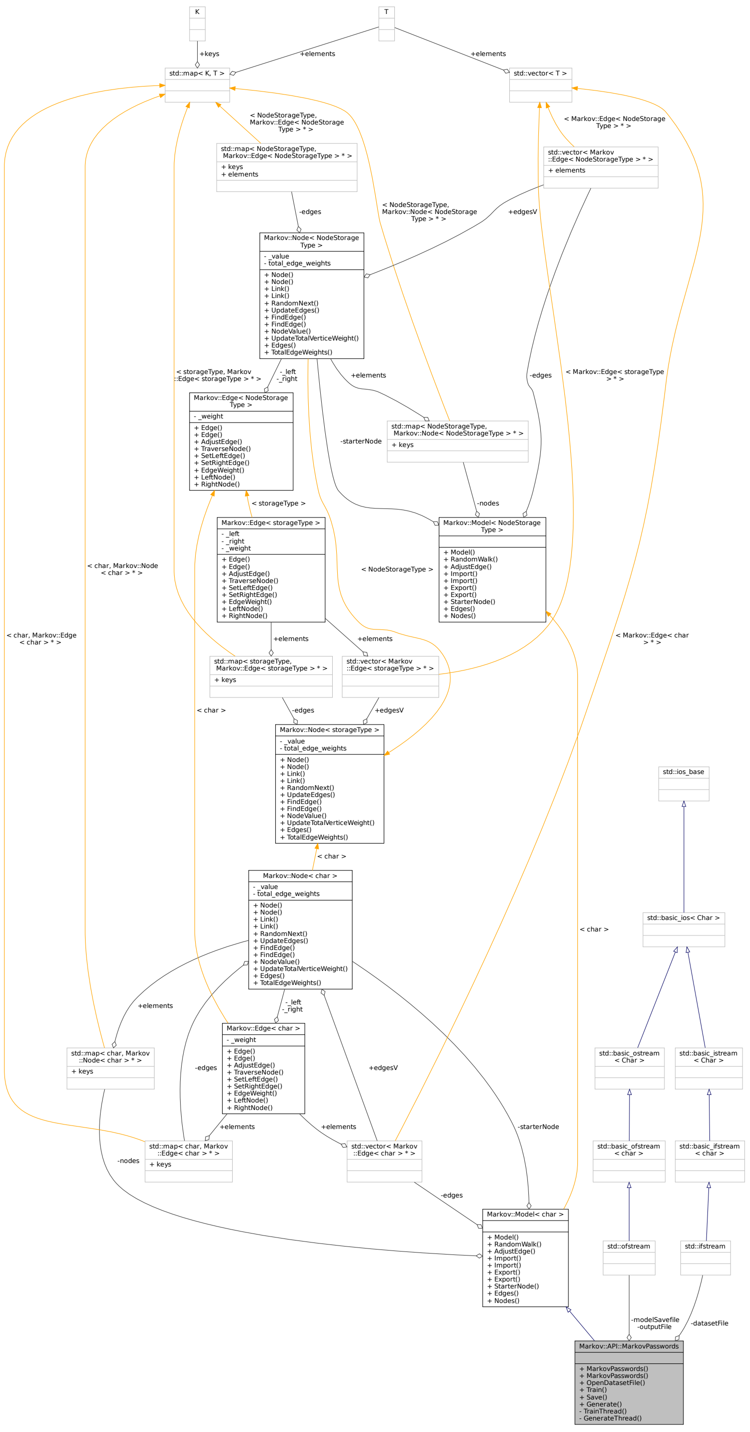
Markov::Model with char represented nodes.

`#include <markovPasswords.h>`

Inheritance diagram for Markov::API::MarkovPasswords:



Collaboration diagram for Markov::API::MarkovPasswords:



Public Member Functions

- [MarkovPasswords](#) ()

  Initialize the markov model from MarkovModel::Markov::Model.
- [MarkovPasswords](#) (const char ∗filename)

  Initialize the markov model from MarkovModel::Markov::Model, with an import file.
- std::ifstream ∗ [OpenDatasetFile](#) (const char ∗filename)

  Open dataset file and return the ifstream pointer.

- void Train (const char ∗datasetFileName, char delimiter, int threads)

  Train the model with the dataset file.
- std::ofstream ∗ Save (const char ∗filename)

  Export model to file.
- void Generate (unsigned long int n, const char ∗wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

  Call Markov::Model::RandomWalk n times, and collect output.
- char ∗ RandomWalk (Markov::Random::RandomEngine ∗randomEngine, int minSetting, int maxSetting, char ∗buffer)

  Do a random walk on this model.
- void AdjustEdge (const char ∗payload, long int occurrence)

  Adjust the model with a single string.
- bool Import (std::ifstream ∗)

  Import a file to construct the model.
- bool Import (const char ∗filename)

  Open a file to import with filename, and call bool Model::Import with std::ifstream.
- bool Export (std::ofstream ∗)

  Export a file of the model.
- bool Export (const char ∗filename)

  Open a file to export with filename, and call bool Model::Export with std::ofstream.
- Node< char > ∗ StarterNode ()

  Return starter Node.
- std::vector< Edge< char > ∗ > ∗ Edges ()

  Return a vector of all the edges in the model.
- std::map< char, Node< char > ∗ > ∗ Nodes ()

  Return starter Node.

Private Member Functions

- void TrainThread (Markov::API::Concurrency::ThreadSharedListHandler ∗listhandler, char delimiter)

  A single thread invoked by the Train function.
- void GenerateThread (std::mutex ∗outputLock, unsigned long int n, std::ofstream ∗wordlist, int minLen, int maxLen)

  A single thread invoked by the Generate function.

Private Attributes

- std::ifstream ∗ datasetFile
- std::ofstream ∗ modelSavefile
- std::ofstream ∗ outputFile
- std::map< char, Node< char > ∗ > nodes

  Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > ∗ starterNode

  Starter Node of this model.
- std::vector< Edge< char > ∗ > edges

  A list of all edges in this model.

8.9.1 Detailed Description

Markov::Model with char represented nodes.
Includes wrappers for Markov::Model and additional helper functions to handle file I/O
This class is an extension of Markov::Model<char>, with higher level abstractions such as train and generate.
Definition at line 17 of file markovPasswords.h.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 MarkovPasswords() [1/2]

`Markov::API::MarkovPasswords::MarkovPasswords ( )`

Initialize the markov model from MarkovModel::Markov::Model.

Parent constructor. Has no extra functionality.

Definition at line 10 of file markovPasswords.cpp.

```
                                              : Markov::Model<char>(){


}
```

8.9.2.2 MarkovPasswords() [2/2]

```
Markov::API::MarkovPasswords::MarkovPasswords (
              const char * filename )
```

Initialize the markov model from MarkovModel::Markov::Model, with an import file.

This function calls the Markov::Model::Import on the filename to construct the model. Same thing as creating and empty model, and calling MarkovPasswords::Import on the filename.

Parameters

filename	- Filename to import

Example Use: Construction via filename

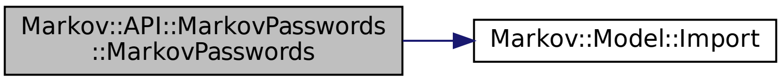
```
MarkovPasswords mp("test.mdl");
```

Definition at line 15 of file markovPasswords.cpp.

```
                                                            {

    std::ifstream* importFile;

    this->Import(filename);

    //std::ifstream* newFile(filename);

    //importFile = newFile;

}
```

References Markov::Model< NodeStorageType >::Import().

Here is the call graph for this function:



8.9.3 Member Function Documentation

8.9.3.1 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
              const char * payload,
              long int occurrence )   [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.
Then, update the edge EdgeWeight from current node, to the terminator node.
This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.
Example Use: Create an empty model and train it with string: "testdata"

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

Parameters

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

Definition at line 322 of file model.h.

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

8.9.3.2 Edges()

`std::vector<Edge<char >*>* Markov::Model< char >::Edges  [inline], [inherited]`

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 172 of file model.h.

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

8.9.3.3 Export() [1/2]

```
bool Markov::Model< char >::Export (
            const char * filename )  [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

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

Definition at line 285 of file model.h.

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

8.9.3.4 Export() [2/2]

```
bool Markov::Model< char >::Export (
            std::ofstream * f ) [inherited]
```

Export a file of the model.
File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.
Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

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

Definition at line 273 of file model.h.

```
                                                   {
    Markov::Edge<NodeStorageType>* e;
    for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
        e = this->edges[i];
        //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
        e->RightNode()->NodeValue() « "\n";
        *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
        "\n";
    }

    return true;
}
```

8.9.3.5 Generate()

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

Call Markov::Model::RandomWalk n times, and collect output.
Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

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

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

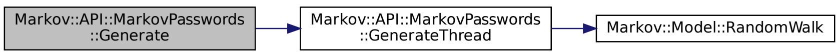
Definition at line 92 of file markovPasswords.cpp.

```

                                   {
    char* res;
    char print[100];
    std::ofstream wordlist;
    wordlist.open(wordlistFileName);
    std::mutex mlock;
    int iterationsPerThread = n/threads;
    int iterationsCarryOver = n%threads;
    std::vector<std::thread*> threadsV;
    for(int i=0;i<threads;i++){
        threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
        &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
    }

    for(int i=0;i<threads;i++){
        threadsV[i]->join();
        delete threadsV[i];
    }

    this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);

}
```

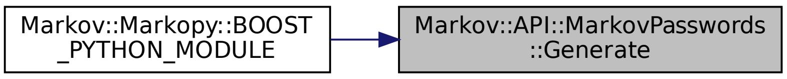
References GenerateThread().

Referenced by Markov::Markopy::BOOST_PYTHON_MODULE().

Here is the call graph for this function:


Markov::API::MarkovPasswords ::Generate
Markov::API::MarkovPasswords ::GenerateThread
Markov::Model::RandomWalk

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::API::MarkovPasswords ::Generate

8.9.3.6 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
            std::mutex * outputLock,
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen,
            int maxLen ) [private]
```

A single thread invoked by the Generate function.

DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Parameters

outputLock	- shared mutex lock to lock during output operation. Prevents race condition on write.
n	number of lines to be generated by this thread
wordlist	wordlistfile
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

Definition at line 114 of file markovPasswords.cpp.

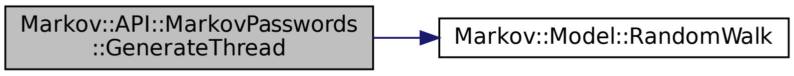
```

                                              {
    char* res = new char[maxLen+5];
    if(n==0) return;

    Markov::Random::Marsaglia MarsagliaRandomEngine;
    for (int i = 0; i < n; i++) {
        this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
```

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

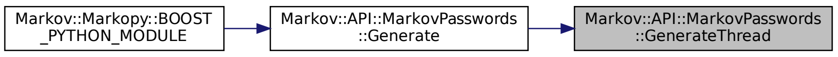
References Markov::Model< NodeStorageType >::RandomWalk().

Referenced by Generate().

Here is the call graph for this function:


Markov::API::MarkovPasswords ::GenerateThread
Markov::Model::RandomWalk

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::API::MarkovPasswords ::Generate
Markov::API::MarkovPasswords ::GenerateThread

8.9.3.7 Import() [1/2]

```
bool Markov::Model< char >::Import (
              const char * filename )  [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

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

Definition at line 265 of file model.h.

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

8.9.3.8 Import() [2/2]

```
bool Markov::Model< char >::Import (
              std::ifstream * f )  [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

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

Definition at line 206 of file model.h.

```
                                                         {
    std::string cell;

    char src;
    char target;
    long int oc;

    while (std::getline(*f, cell)) {
        //std::cout « "cell: " « cell « std::endl;
        src = cell[0];
        target = cell[cell.length() - 1];
        char* j;
        oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
        //std::cout « oc « "\n";
        Markov::Node<NodeStorageType>* srcN;
        Markov::Node<NodeStorageType>* targetN;
        Markov::Edge<NodeStorageType>* e;
        if (this->nodes.find(src) == this->nodes.end()) {
            srcN = new Markov::Node<NodeStorageType>(src);
            this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
            //std::cout « "Creating new node at start.\n";
        }
        else {
            srcN = this->nodes.find(src)->second;
        }

        if (this->nodes.find(target) == this->nodes.end()) {
            targetN = new Markov::Node<NodeStorageType>(target);
            this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
            //std::cout « "Creating new node at end.\n";
        }
        else {
            targetN = this->nodes.find(target)->second;
        }
        e = srcN->Link(targetN);
        e->AdjustEdge(oc);
        this->edges.push_back(e);

        //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
    int(targetN->NodeValue()) « "\n";


    }

    for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
        //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
        std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
    Edge<NodeStorageType> *rhs)->bool{
            return lhs->EdgeWeight() > rhs->EdgeWeight();
        });
        //for(int i=0;i<x.second->edgesV.size();i++)
        //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
        //std::cout « "\n";
    }
    //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
    //std::cout « "Total number of edges: " « this->edges.size() « std::endl;

    return true;
}
```

8.9.3.9 Nodes()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 177 of file model.h.

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

8.9.3.10 OpenDatasetFile()

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

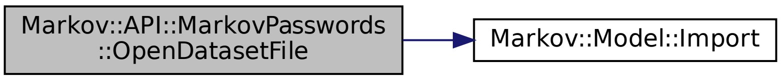
ifstream∗ to the the dataset file

Definition at line 27 of file markovPasswords.cpp.

```
                                                                                {

    std::ifstream* datasetFile;

    std::ifstream newFile(filename);

    datasetFile = &newFile;

    this->Import(datasetFile);
    return datasetFile;
}
```

References Markov::Model< NodeStorageType >::Import().

Here is the call graph for this function:



8.9.3.11 RandomWalk()

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 292 of file model.h.

```
                                                      {
    Markov::Node<NodeStorageType>* n = this->starterNode;
    int len = 0;
    Markov::Node<NodeStorageType>* 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();
    }

    //null terminate the string
    buffer[len] = 0x00;

    //do something with the generated string
    return buffer; //for now
}
```

8.9.3.12 Save()

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

Export model to file.

Parameters

filename	- Export filename.

Returns

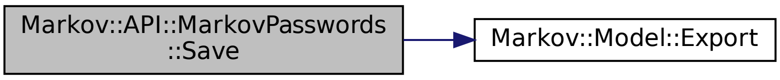
std::ofstream∗ of the exported file.

Definition at line 80 of file markovPasswords.cpp.

```
                                                                   {
    std::ofstream* exportFile;

    std::ofstream newFile(filename);

    exportFile = &newFile;

    this->Export(exportFile);
    return exportFile;
}
```

References Markov::Model< NodeStorageType >::Export().

Here is the call graph for this function:



8.9.3.13 StarterNode()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 167 of file model.h.

```
{ return starterNode;}
```

8.9.3.14 Train()

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

Train the model with the dataset file.

Parameters

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

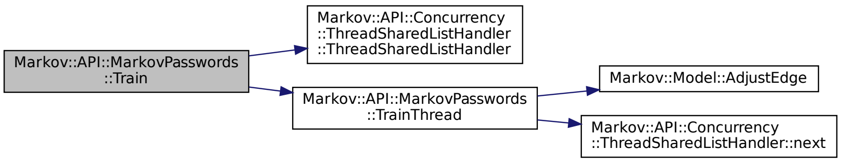
```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

Definition at line 40 of file markovPasswords.cpp.

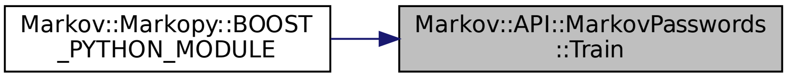
```
                                                                                      {
    Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
    auto start = std::chrono::high_resolution_clock::now();

    std::vector<std::thread*> threadsV;
    for(int i=0;i<threads;i++){
        threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
    &listhandler, 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";

}
```

References Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler(), and TrainThread().

Referenced by Markov::Markopy::BOOST_PYTHON_MODULE().
Here is the call graph for this function:


Markov::API::MarkovPasswords ::Train
Markov::API::Concurrency ::ThreadSharedListHandler ::ThreadSharedListHandler
Markov::API::MarkovPasswords ::TrainThread
Markov::Model::AdjustEdge
Markov::API::Concurrency ::ThreadSharedListHandler::next

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::API::MarkovPasswords ::Train

8.9.3.15 TrainThread()

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

A single thread invoked by the Train function.

Parameters

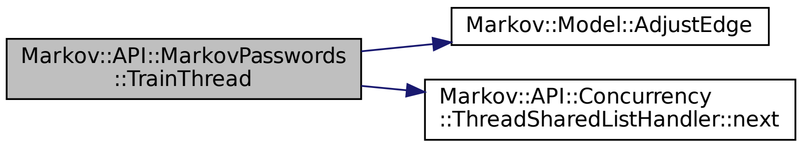
listhandler	- Listhandler class to read corpus from
delimiter	- a character, same as the delimiter in dataset content

Definition at line 59 of file markovPasswords.cpp.

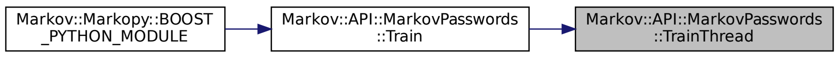
```
                    {
    char format_str[] ="%ld,%s";
    format_str[2]=delimiter;
    std::string line;
    while (listhandler->next(&line)) {
        long int oc;
        if (line.size() > 100) {
            line = line.substr(0, 100);
        }
        char* linebuf = new char[line.length()+5];
#ifdef _WIN32
        sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
#else
        sscanf(line.c_str(), format_str, &oc, linebuf);
#endif
        this->AdjustEdge((const char*)linebuf, oc);
        delete linebuf;
    }
}
```

References Markov::Model< NodeStorageType >::AdjustEdge(), and Markov::API::Concurrency::ThreadSharedListHandler::next().
Referenced by Train().

Here is the call graph for this function:


Markov::API::MarkovPasswords ::TrainThread
Markov::Model::AdjustEdge
Markov::API::Concurrency ::ThreadSharedListHandler::next

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::API::MarkovPasswords ::Train
Markov::API::MarkovPasswords ::TrainThread

8.9.4 Member Data Documentation

8.9.4.1 datasetFile

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

Definition at line 106 of file markovPasswords.h.

8.9.4.2 edges

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

A list of all edges in this model.

Definition at line 194 of file model.h.

8.9.4.3 modelSavefile

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

Definition at line 107 of file markovPasswords.h.

8.9.4.4 nodes

`std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]`

Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.

Definition at line 183 of file model.h.

8.9.4.5 outputFile

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

Definition at line 108 of file markovPasswords.h.

8.9.4.6 starterNode

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

Starter Node of this model.

Definition at line 188 of file model.h.

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

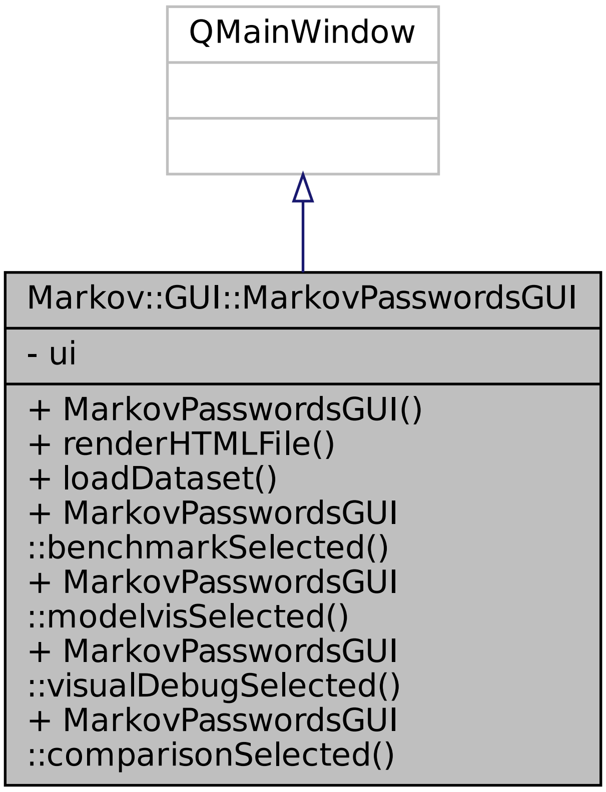
- markovPasswords.h
- markovPasswords.cpp

8.10 Markov::GUI::MarkovPasswordsGUI Class Reference

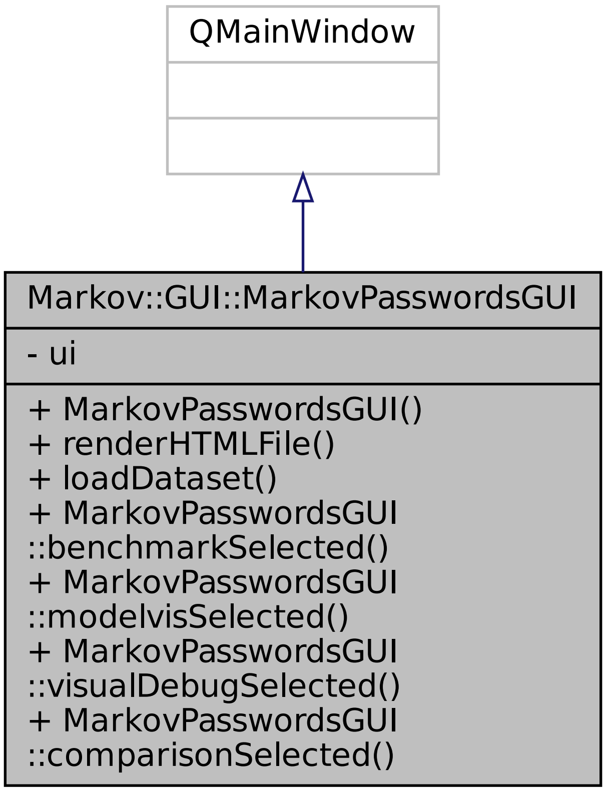
Reporting UI.

`#include <MarkovPasswordsGUI.h>`

Inheritance diagram for Markov::GUI::MarkovPasswordsGUI:


QMainWindow
Markov::GUI::MarkovPasswordsGUI
- ui
+ MarkovPasswordsGUI()
+ renderHTMLFile()
+ loadDataset()
+ MarkovPasswordsGUI::benchmarkSelected()
+ MarkovPasswordsGUI::modelvisSelected()
+ MarkovPasswordsGUI::visualDebugSelected()
+ MarkovPasswordsGUI::comparisonSelected()

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 consturctor.
- void renderHTMLFile (std::string ∗filename)

  Render a HTML file.
- void loadDataset (std::string ∗filename)

  Load a dataset to current view..

Private Attributes

- Ui::MarkovPasswordsGUIClass ui

8.10.1 Detailed Description

Reporting UI.
UI for reporting and debugging tools for MarkovPassword
Definition at line 12 of file MarkovPasswordsGUI.h.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 MarkovPasswordsGUI()

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

Default QT consturctor.

Parameters

parent	- Parent widget.

Definition at line 8 of file MarkovPasswordsGUI.cpp.

```
    : QMainWindow(parent)
{
    ui.setupUi(this);


    QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {benchmarkSelected(); });
    QObject::connect(ui.pushButton_2,&QPushButton::clicked, this, [this] {modelvisSelected(); });
    QObject::connect(ui.pushButton_4, &QPushButton::clicked, this, [this] {comparisonSelected(); });
}
```

8.10.3 Member Function Documentation

8.10.3.1 loadDataset()

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

Load a dataset to current view..

Parameters

filename	- Filename of the dataset file. (relative path to the views folder).

Definition at line 78 of file MarkovPasswordsGUI.cpp.

```
                                                       {
    //extract and parametrize the code from constructor

}
```

8.10.3.2 MarkovPasswordsGUI::benchmarkSelected

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

8.10.3.3 MarkovPasswordsGUI::comparisonSelected

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

8.10.3.4 MarkovPasswordsGUI::modelvisSelected

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

8.10.3.5 MarkovPasswordsGUI::visualDebugSelected

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

8.10.3.6 renderHTMLFile()

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

Render a HTML file.

Parameters

filename	- Filename of the html file. (relative path to the views folder).

Definition at line 71 of file MarkovPasswordsGUI.cpp.

```
                                                           {
    //extract and parametrize the code from constructor

}
```

8.10.4 Member Data Documentation

8.10.4.1 ui

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

Definition at line 32 of file MarkovPasswordsGUI.h.

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

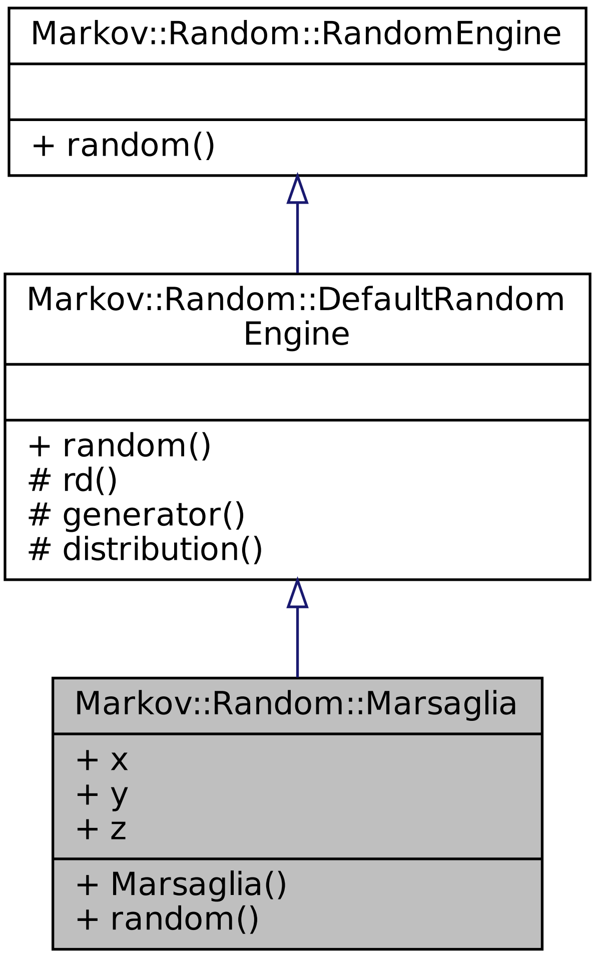
- MarkovPasswordsGUI.h
- MarkovPasswordsGUI.cpp

8.11 Markov::Random::Marsaglia Class Reference

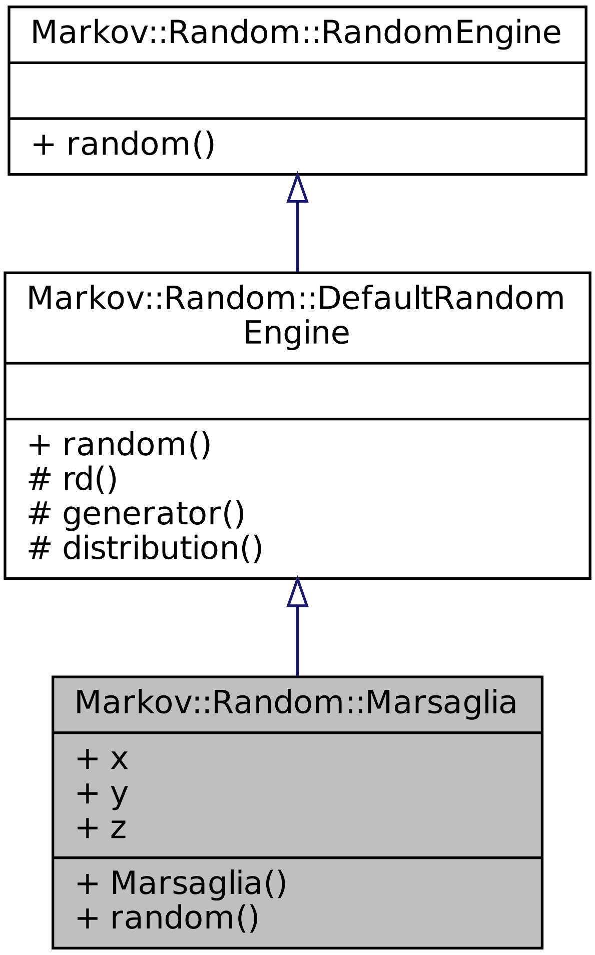
Implementation of Marsaglia Random Engine.

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

Inheritance diagram for Markov::Random::Marsaglia:


Markov::Random::RandomEngine
+ random()
Markov::Random::DefaultRandom Engine
+ random()
# rd()
# generator()
# distribution()
Markov::Random::Marsaglia
+ x
+ y
+ z
+ Marsaglia()
+ random()

Collaboration diagram for Markov::Random::Marsaglia:


Markov::Random::RandomEngine
+ random()
Markov::Random::DefaultRandom Engine
+ random()
# rd()
# generator()
# distribution()
Markov::Random::Marsaglia
+ x
+ y
+ z
+ Marsaglia()
+ random()

Public Member Functions

- Marsaglia ()

  Construct Marsaglia Engine.
- unsigned long random ()

  Generate Random Number.

Public Attributes

- unsigned long x
- unsigned long y
- unsigned long z

Protected Member Functions

- std::random_device & rd ()

  Default random device for seeding.

- std::default_random_engine & generator ()

  Default random engine for seeding.
- std::uniform_int_distribution< long long unsigned > & distribution ()

  Distribution schema for seeding.

8.11.1 Detailed Description

Implementation of Marsaglia Random Engine.

This is an implementation of Marsaglia Random engine, which for most use cases is a better fit than other solutions. Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.

This implementation of the Marsaglia Engine is seeded by random.h default random engine. RandomEngine is only seeded once so its not a performance issue.

Example Use: Using Marsaglia Engine with RandomWalk

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

Example Use: Generating a random number with Marsaglia Engine

```
Markov::Random::Marsaglia me;
std::cout « me.random();
```

Definition at line 116 of file random.h.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 Marsaglia()

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

Construct Marsaglia Engine.

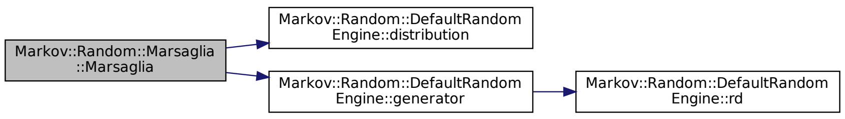
Initialize x,y and z using the default random engine.

Definition at line 123 of file random.h.

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

References Markov::Random::DefaultRandomEngine::distribution(), Markov::Random::DefaultRandomEngine::generator(), x, y, and z.

Here is the call graph for this function:


Markov::Random::Marsaglia ::Marsaglia
Markov::Random::DefaultRandom Engine::distribution
Markov::Random::DefaultRandom Engine::generator
Markov::Random::DefaultRandom Engine::rd

8.11.3 Member Function Documentation

8.11.3.1 distribution()

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

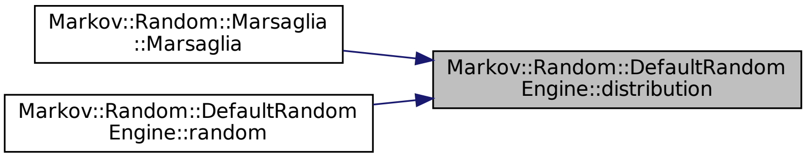
Distribution schema for seeding.

Definition at line 81 of file random.h.

```
                                                                                {
            static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffFFFF);
            return _distribution;
        }
```

Referenced by Marsaglia(), and Markov::Random::DefaultRandomEngine::random().

Here is the caller graph for this function:


Markov::Random::Marsaglia ::Marsaglia
Markov::Random::DefaultRandom Engine::random
Markov::Random::DefaultRandom Engine::distribution

8.11.3.2 generator()

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

Default random engine for seeding.

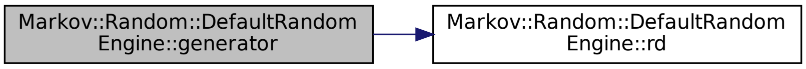
Definition at line 73 of file random.h.

```
                                                    {
            static std::default_random_engine _generator(rd()());
            return _generator;
        }
```

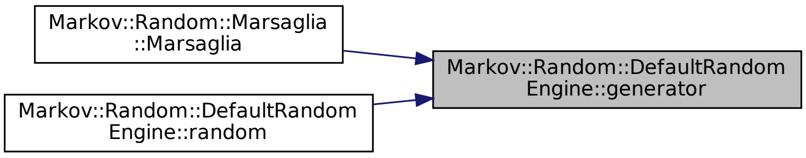
References Markov::Random::DefaultRandomEngine::rd().

Referenced by Marsaglia(), and Markov::Random::DefaultRandomEngine::random().

Here is the call graph for this function:


Markov::Random::DefaultRandom Engine::generator
Markov::Random::DefaultRandom Engine::rd

Here is the caller graph for this function:


Markov::Random::Marsaglia ::Marsaglia
Markov::Random::DefaultRandom Engine::random
Markov::Random::DefaultRandom Engine::generator

8.11.3.3 random()

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

Generate Random Number.

Returns

random number in long range.

Reimplemented from Markov::Random::DefaultRandomEngine.

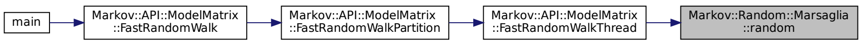
Definition at line 131 of file random.h.

```
                              {
        unsigned long t;
        x ^= x « 16;
        x ^= x » 5;
        x ^= x « 1;

        t = x;
        x = y;
        y = z;
        z = t ^ x ^ y;

        return z;
    }
```

References x, y, and z.

Referenced by Markov::API::ModelMatrix::FastRandomWalkThread().

Here is the caller graph for this function:


main
Markov::API::ModelMatrix ::FastRandomWalk
Markov::API::ModelMatrix ::FastRandomWalkPartition
Markov::API::ModelMatrix ::FastRandomWalkThread
Markov::Random::Marsaglia ::random

8.11.3.4 rd()

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

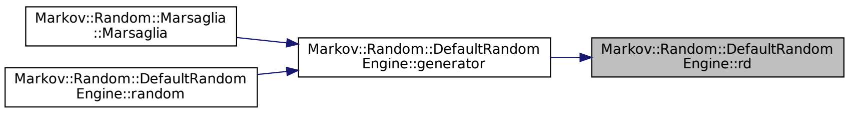
Default random device for seeding.

Definition at line 65 of file random.h.

```
                              {
            static std::random_device _rd;
            return _rd;
        }
```

Referenced by Markov::Random::DefaultRandomEngine::generator().

Here is the caller graph for this function:


Markov::Random::Marsaglia ::Marsaglia
Markov::Random::DefaultRandom Engine::random
Markov::Random::DefaultRandom Engine::generator
Markov::Random::DefaultRandom Engine::rd

8.11.4 Member Data Documentation

8.11.4.1 x

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

Definition at line 146 of file random.h.

Referenced by Marsaglia(), and random().

8.11.4.2 y

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

Definition at line 147 of file random.h.

Referenced by Marsaglia(), and random().

8.11.4.3 z

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

Definition at line 148 of file random.h.

Referenced by Marsaglia(), and random().

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

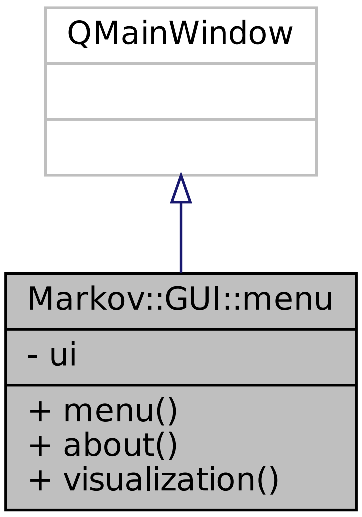
- random.h

8.12 Markov::GUI::menu Class Reference

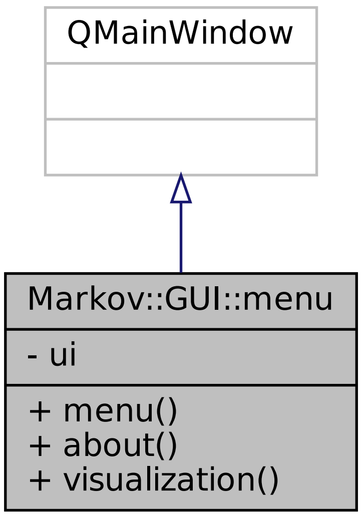
QT Menu class.

`#include <menu.h>`

Inheritance diagram for Markov::GUI::menu:



Collaboration diagram for Markov::GUI::menu:



Public Slots

- void about ()
- void visualization ()

Public Member Functions

- menu (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::main ui

8.12.1 Detailed Description

QT Menu class.
Definition at line 9 of file menu.h.

8.12.2 Constructor & Destructor Documentation

8.12.2.1 menu()

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

Definition at line 8 of file menu.cpp.

```
    : QMainWindow(parent)
{
    ui.setupUi(this);


    //QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {about(); });
    QObject::connect(ui.visu, &QPushButton::clicked, this, [this] {visualization(); });
}
```

8.12.3 Member Function Documentation

8.12.3.1 about

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

Definition at line 17 of file menu.cpp.

```
                    {


}
```

8.12.3.2 visualization

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

Definition at line 21 of file menu.cpp.

```
                            {
    MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
    w->show();
    this->close();
}
```

8.12.4 Member Data Documentation

8.12.4.1 ui

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

Definition at line 15 of file menu.h.

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

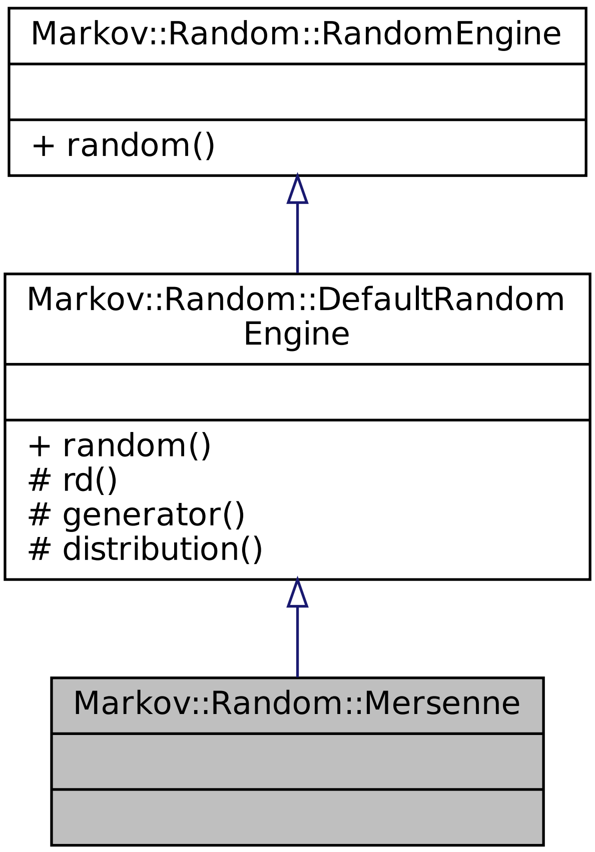
- menu.h
- menu.cpp

8.13 Markov::Random::Mersenne Class Reference

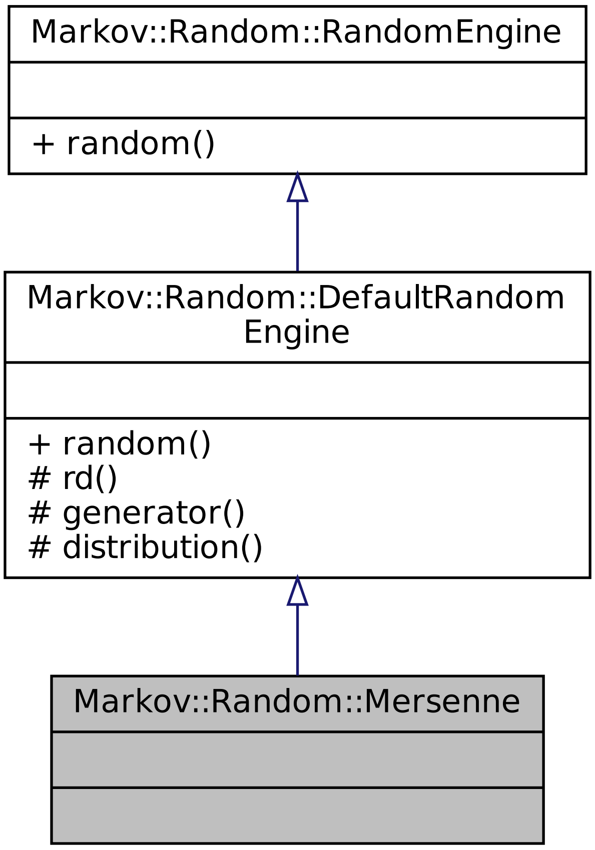
Implementation of Mersenne Twister Engine.

`#include <random.h>`

Inheritance diagram for Markov::Random::Mersenne:



Collaboration diagram for Markov::Random::Mersenne:



Public Member Functions

- unsigned long random ()

  Generate Random Number.

Protected Member Functions

- std::random_device & rd ()

  Default random device for seeding.
- std::default_random_engine & generator ()

  Default random engine for seeding.
- std::uniform_int_distribution< long long unsigned > & distribution ()

  Distribution schema for seeding.

8.13.1 Detailed Description

Implementation of Mersenne Twister Engine.
This is an implementation of Mersenne Twister Engine, which is slow but is a good implementation for high entropy pseudorandom.
Example Use: Using Mersenne Engine with RandomWalk

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

Example Use: Generating a random number with Marsaglia Engine

```
Markov::Random::Mersenne me;
std::cout « me.random();
```

Definition at line 176 of file random.h.

8.13.2 Member Function Documentation

8.13.2.1 distribution()

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

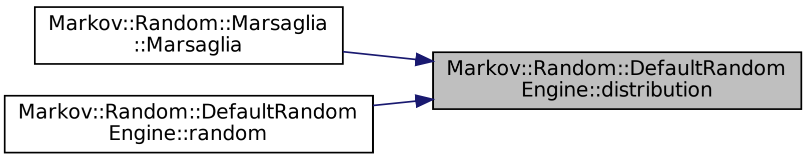
Distribution schema for seeding.

Definition at line 81 of file random.h.

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

Referenced by Markov::Random::Marsaglia::Marsaglia(), and Markov::Random::DefaultRandomEngine::random().

Here is the caller graph for this function:


Markov::Random::Marsaglia ::Marsaglia
Markov::Random::DefaultRandom Engine::random
Markov::Random::DefaultRandom Engine::distribution

8.13.2.2 generator()

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

Default random engine for seeding.

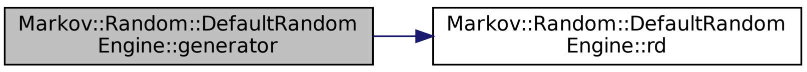
Definition at line 73 of file random.h.

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

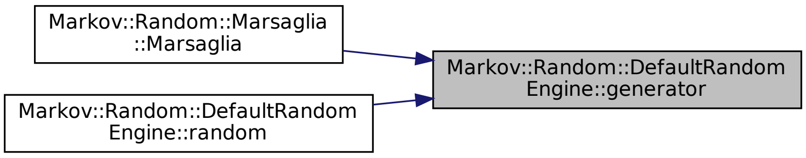
References Markov::Random::DefaultRandomEngine::rd().

Referenced by Markov::Random::Marsaglia::Marsaglia(), and Markov::Random::DefaultRandomEngine::random().

Here is the call graph for this function:


Markov::Random::DefaultRandom Engine::generator
Markov::Random::DefaultRandom Engine::rd

Here is the caller graph for this function:


Markov::Random::Marsaglia ::Marsaglia
Markov::Random::DefaultRandom Engine::random
Markov::Random::DefaultRandom Engine::generator

8.13.2.3 random()

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

Generate Random Number.

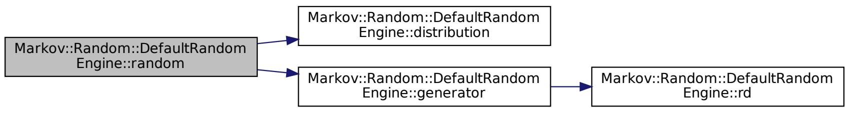
Returns

random number in long range.

Implements Markov::Random::RandomEngine.
Reimplemented in Markov::Random::Marsaglia.
Definition at line 57 of file random.h.

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

References Markov::Random::DefaultRandomEngine::distribution(), and Markov::Random::DefaultRandomEngine::generator().
Here is the call graph for this function:


Markov::Random::DefaultRandom Engine::random
Markov::Random::DefaultRandom Engine::distribution
Markov::Random::DefaultRandom Engine::generator
Markov::Random::DefaultRandom Engine::rd

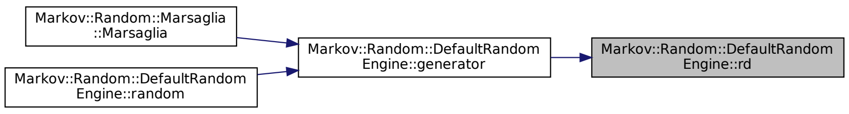
8.13.2.4 rd()

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

Default random device for seeding.
Definition at line 65 of file random.h.

```
                                {
            static std::random_device _rd;
            return _rd;
        }
```

Referenced by Markov::Random::DefaultRandomEngine::generator().
Here is the caller graph for this function:


Markov::Random::Marsaglia ::Marsaglia
Markov::Random::DefaultRandom Engine::random
Markov::Random::DefaultRandom Engine::generator
Markov::Random::DefaultRandom Engine::rd

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

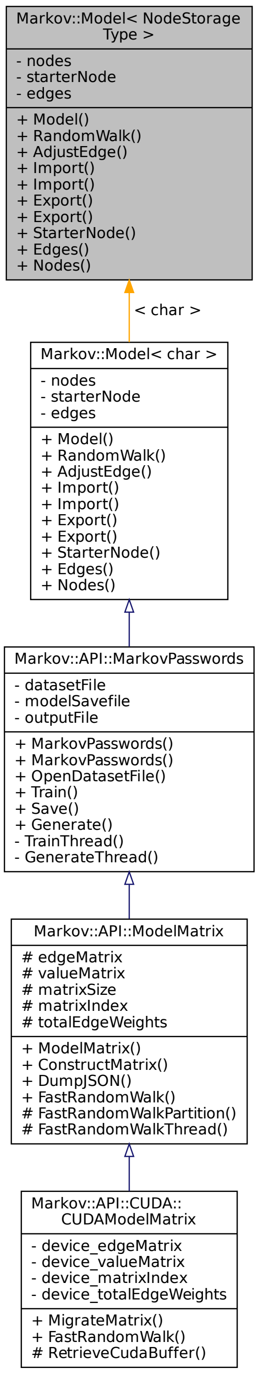
- random.h

8.14 Markov::Model< NodeStorageType > Class Template Reference

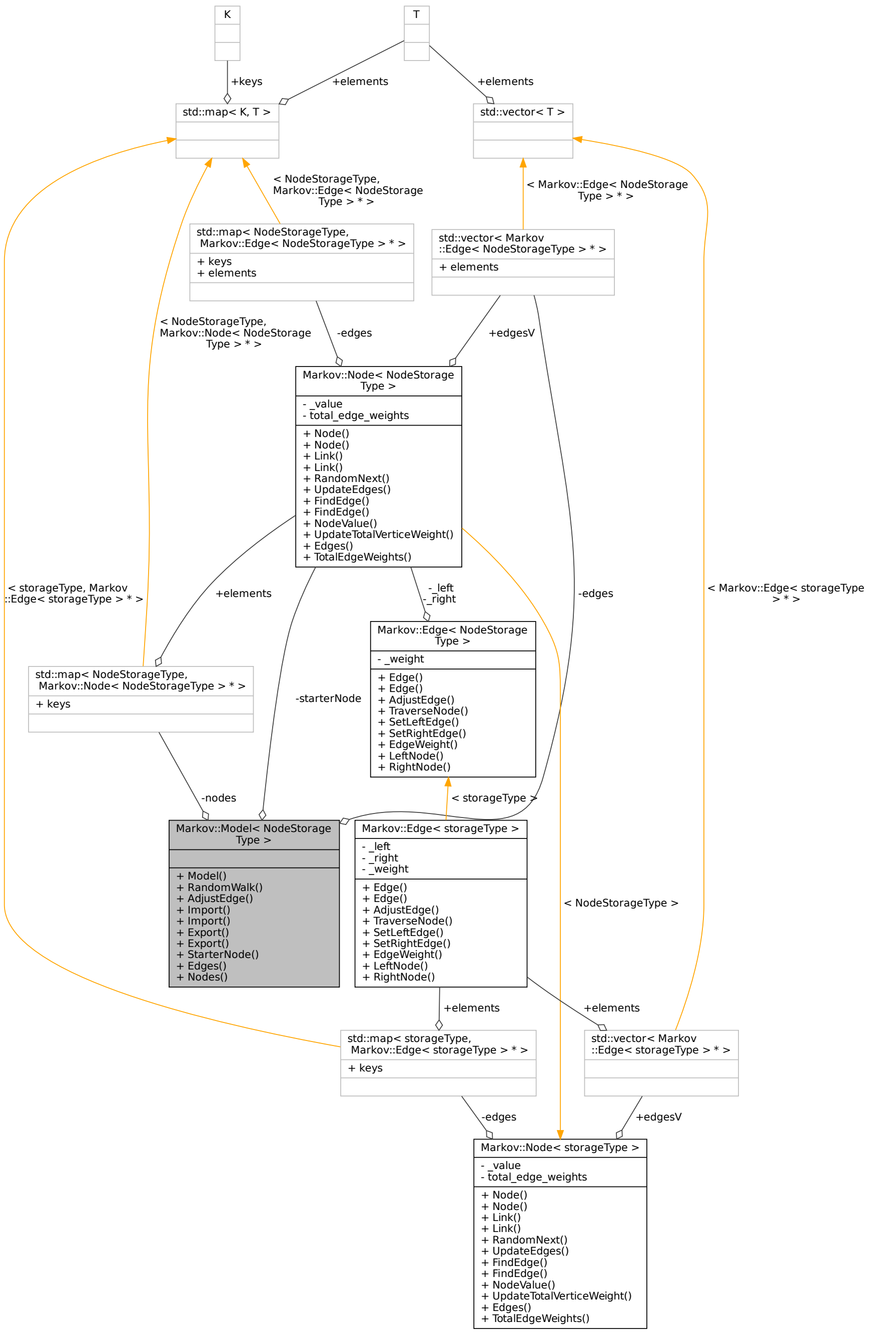
class for the final Markov Model, constructed from nodes and edges.

`#include <model.h>`

Inheritance diagram for Markov::Model< NodeStorageType >:



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.

8.14.1 Detailed Description

template<typename NodeStorageType>
class Markov::Model< NodeStorageType >

class for the final Markov Model, constructed from nodes and edges.
Each atomic piece of the generation result is stored in a node, while edges contain the relation weights. *Extending:* To extend the class, implement the template and inherit from it, as "class MyModel : public Markov::Model<char>". For a complete demonstration of how to extend the class, see MarkovPasswords.
Whole model can be defined as a list of the edges, as dangling nodes are pointless. This approach is used for the import/export operations. For more information on importing/exporting model, check out the github readme and wiki page.
Definition at line 41 of file model.h.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 Model()

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

Initialize a model with only start and end nodes.
Initialize an empty model with only a starterNode Starter node is a special kind of node that has constant 0x00 value, and will be used to initiate the generation execution from.
Definition at line 200 of file model.h.

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

8.14.3 Member Function Documentation

8.14.3.1 AdjustEdge()

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

Adjust the model with a single string.
Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.
Then, update the edge EdgeWeight from current node, to the terminator node.
This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.
Example Use: Create an empty model and train it with string: "testdata"

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

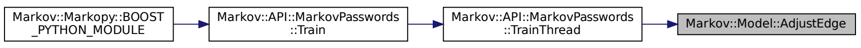
Parameters

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

Definition at line 322 of file model.h.

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

Referenced by Markov::API::MarkovPasswords::TrainThread().
Here is the caller graph for this function:


Markov::Markopy::BOOST_PYTHON_MODULE
Markov::API::MarkovPasswords::Train
Markov::API::MarkovPasswords::TrainThread
Markov::Model::AdjustEdge

8.14.3.2 Edges()

```
template<typename NodeStorageType >
std::vector<Edge<NodeStorageType>*>* Markov::Model< NodeStorageType >::Edges ( )  [inline]
```

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 172 of file model.h.

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

8.14.3.3 Export() [1/2]

```
template<typename NodeStorageType >
bool Markov::Model< NodeStorageType >::Export (
             const char * filename )
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

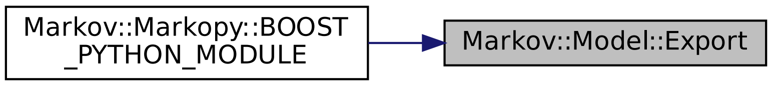
```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 285 of file model.h.

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

Referenced by Markov::Markopy::BOOST_PYTHON_MODULE().

Here is the caller graph for this function:



8.14.3.4 Export() [2/2]

```
template<typename NodeStorageType >
bool Markov::Model< NodeStorageType >::Export (
             std::ofstream * f )
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

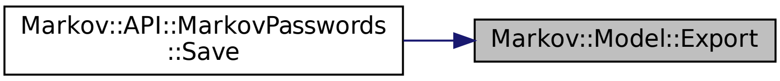
```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 273 of file model.h.

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

Referenced by Markov::API::MarkovPasswords::Save().

Here is the caller graph for this function:



8.14.3.5 Import() [1/2]

```
template<typename NodeStorageType >
bool Markov::Model< NodeStorageType >::Import (
            const char * filename )
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

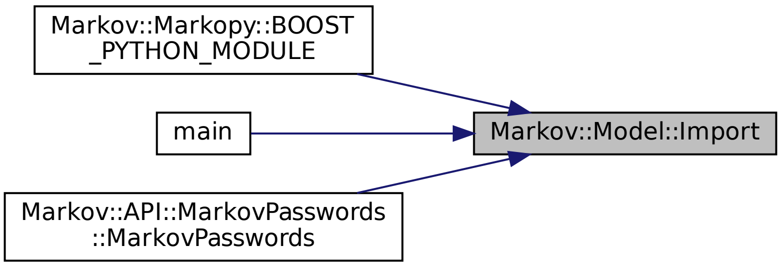
```
Markov::Model<char> model;
model.Import("test.mdl");
```

Definition at line 265 of file model.h.

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

Referenced by Markov::Markopy::BOOST_PYTHON_MODULE(), main(), and Markov::API::MarkovPasswords::MarkovPasswords().

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
main
Markov::API::MarkovPasswords ::MarkovPasswords
Markov::Model::Import

8.14.3.6 Import() [2/2]

```
template<typename NodeStorageType >
bool Markov::Model< NodeStorageType >::Import (
            std::ifstream * f )
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

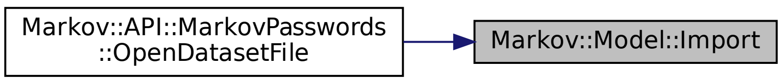
```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 206 of file model.h.

```
                                                              {
    std::string cell;

    char src;
    char target;
    long int oc;

    while (std::getline(*f, cell)) {
        //std::cout « "cell: " « cell « std::endl;
        src = cell[0];
        target = cell[cell.length() - 1];
        char* j;
        oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
        //std::cout « oc « "\n";
        Markov::Node<NodeStorageType>* srcN;
        Markov::Node<NodeStorageType>* targetN;
        Markov::Edge<NodeStorageType>* e;
        if (this->nodes.find(src) == this->nodes.end()) {
            srcN = new Markov::Node<NodeStorageType>(src);
            this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
            //std::cout « "Creating new node at start.\n";
        }
        else {
            srcN = this->nodes.find(src)->second;
        }

        if (this->nodes.find(target) == this->nodes.end()) {
            targetN = new Markov::Node<NodeStorageType>(target);
            this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
            //std::cout « "Creating new node at end.\n";
        }
        else {
            targetN = this->nodes.find(target)->second;
        }
        e = srcN->Link(targetN);
        e->AdjustEdge(oc);
        this->edges.push_back(e);

        //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
       int(targetN->NodeValue()) « "\n";


    }

    for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
        //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
        std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
       Edge<NodeStorageType> *rhs)->bool{
            return lhs->EdgeWeight() > rhs->EdgeWeight();
        });
        //for(int i=0;i<x.second->edgesV.size();i++)
        //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
        //std::cout « "\n";
    }
    //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
    //std::cout « "Total number of edges: " « this->edges.size() « std::endl;

    return true;
}
```

Referenced by Markov::API::MarkovPasswords::OpenDatasetFile().

Here is the caller graph for this function:



8.14.3.7 Nodes()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 177 of file model.h.

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

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

Here is the caller graph for this function:

main
Markov::API::ModelMatrix ::ConstructMatrix
Markov::Model::Nodes

8.14.3.8 RandomWalk()

```
template<typename NodeStorageType >
NodeStorageType * Markov::Model< NodeStorageType >::RandomWalk (
                Markov::Random::RandomEngine * randomEngine,
                int minSetting,
                int maxSetting,
                NodeStorageType * buffer )
```

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Parameters

maxSetting	Maximum number of character to generate
buffer	buffer to write the result to

Returns

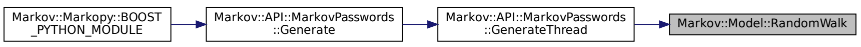
Null terminated string that was generated.

Definition at line 292 of file model.h.

```
                                                          {
    Markov::Node<NodeStorageType>* n = this->starterNode;
    int len = 0;
    Markov::Node<NodeStorageType>* 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();
    }

    //null terminate the string
    buffer[len] = 0x00;

    //do something with the generated string
    return buffer; //for now
}
```

Referenced by Markov::API::MarkovPasswords::GenerateThread().

Here is the caller graph for this function:


Markov::Markopy::BOOST_PYTHON_MODULE
Markov::API::MarkovPasswords::Generate
Markov::API::MarkovPasswords::GenerateThread
Markov::Model::RandomWalk

8.14.3.9 StarterNode()

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

Return starter Node.

Returns

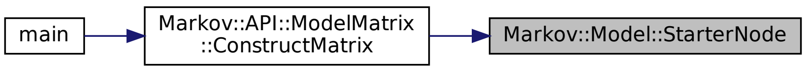
starter node with 00 NodeValue

Definition at line 167 of file model.h.

```
{ return starterNode;}
```

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

Here is the caller graph for this function:


main
Markov::API::ModelMatrix::ConstructMatrix
Markov::Model::StarterNode

8.14.4 Member Data Documentation

8.14.4.1 edges

```
template<typename NodeStorageType >
std::vector<Edge<NodeStorageType>*> Markov::Model< NodeStorageType >::edges  [private]
```
A list of all edges in this model.
Definition at line 194 of file model.h.
Referenced by Markov::Model< char >::Edges().

8.14.4.2 nodes

```
template<typename NodeStorageType >
std::map<NodeStorageType, Node<NodeStorageType>*> Markov::Model< NodeStorageType >::nodes
[private]
```
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 183 of file model.h.
Referenced by Markov::Model< char >::Nodes().

8.14.4.3 starterNode

```
template<typename NodeStorageType >
Node<NodeStorageType>* Markov::Model< NodeStorageType >::starterNode  [private]
```
Starter Node of this model.
Definition at line 188 of file model.h.
Referenced by Markov::Model< char >::StarterNode().
The documentation for this class was generated from the following file:

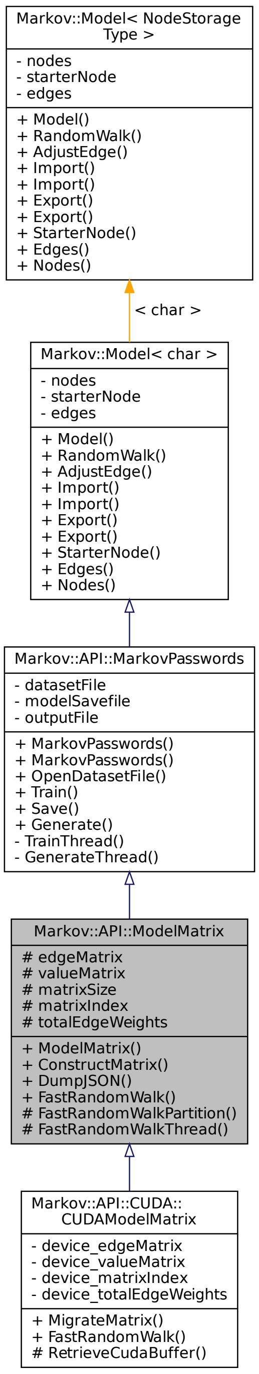
- model.h

8.15 Markov::API::ModelMatrix Class Reference

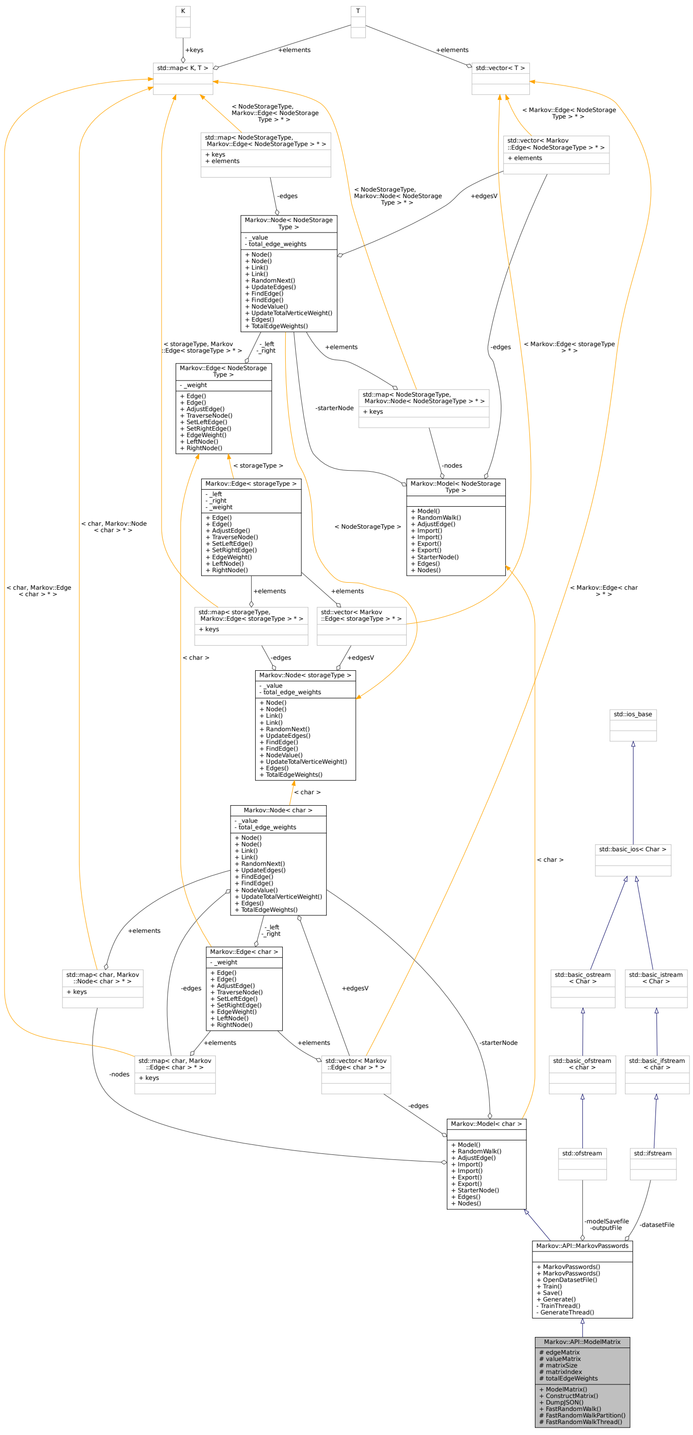
Class to flatten and reduce Markov::Model to a Matrix.

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

Inheritance diagram for Markov::API::ModelMatrix:



Collaboration diagram for Markov::API::ModelMatrix:



Public Member Functions

- ModelMatrix ()
- void ConstructMatrix ()

  Construct the related Matrix data for the model.
- void DumpJSON ()

  Debug function to dump the model to a JSON file.

- void FastRandomWalk (unsigned long int n, const char ∗wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

    Random walk on the Matrix-reduced Markov::Model.
- std::ifstream ∗ OpenDatasetFile (const char ∗filename)

    Open dataset file and return the ifstream pointer.
- void Train (const char ∗datasetFileName, char delimiter, int threads)

    Train the model with the dataset file.
- std::ofstream ∗ Save (const char ∗filename)

    Export model to file.
- void Generate (unsigned long int n, const char ∗wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

    Call Markov::Model::RandomWalk n times, and collect output.
- char ∗ RandomWalk (Markov::Random::RandomEngine ∗randomEngine, int minSetting, int maxSetting, char ∗buffer)

    Do a random walk on this model.
- void AdjustEdge (const char ∗payload, long int occurrence)

    Adjust the model with a single string.
- bool Import (std::ifstream ∗)

    Import a file to construct the model.
- bool Import (const char ∗filename)

    Open a file to import with filename, and call bool Model::Import with std::ifstream.
- bool Export (std::ofstream ∗)

    Export a file of the model.
- bool Export (const char ∗filename)

    Open a file to export with filename, and call bool Model::Export with std::ofstream.
- Node< char > ∗ StarterNode ()

    Return starter Node.
- std::vector< Edge< char > ∗ > ∗ Edges ()

    Return a vector of all the edges in the model.
- std::map< char, Node< char > ∗ > ∗ Nodes ()

    Return starter Node.

Protected Member Functions

- void FastRandomWalkPartition (std::mutex ∗mlock, std::ofstream ∗wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

    A single partition of FastRandomWalk event.
- void FastRandomWalkThread (std::mutex ∗mlock, std::ofstream ∗wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)

    A single thread of a single partition of FastRandomWalk.

Protected Attributes

- char ∗∗ edgeMatrix
- long int ∗∗ valueMatrix
- int matrixSize
- char ∗ matrixIndex
- long int ∗ totalEdgeWeights

Private Member Functions

- void TrainThread (Markov::API::Concurrency::ThreadSharedListHandler ∗listhandler, char delimiter)

  A single thread invoked by the Train function.
- void GenerateThread (std::mutex ∗outputLock, unsigned long int n, std::ofstream ∗wordlist, int minLen, int maxLen)

  A single thread invoked by the Generate function.

Private Attributes

- std::ifstream ∗ datasetFile
- std::ofstream ∗ modelSavefile
- std::ofstream ∗ outputFile
- std::map< char, Node< char > ∗ > nodes

  Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > ∗ starterNode

  Starter Node of this model.
- std::vector< Edge< char > ∗ > edges

  A list of all edges in this model.

8.15.1 Detailed Description

Class to flatten and reduce Markov::Model to a Matrix.
Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)
To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are sychronized and files are flushed every 50M operations.
Definition at line 13 of file modelMatrix.h.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 ModelMatrix()

```
Markov::API::ModelMatrix::ModelMatrix ( )
```

Definition at line 6 of file modelMatrix.cpp.

```
00006                                   {
00007
00008 }
```

8.15.3 Member Function Documentation

8.15.3.1 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
            const NodeStorageType ∗ payload,
            long int occurrence )  [inherited]
```

Adjust the model with a single string.
Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.
Then, update the edge EdgeWeight from current node, to the terminator node.
This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.
Example Use: Create an empty model and train it with string: "testdata"

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

Parameters

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

Definition at line 322 of file model.h.

```
                                                                                {
    NodeStorageType p = payload[0];
    Markov::Node<NodeStorageType>* curnode = this->starterNode;
    Markov::Edge<NodeStorageType>* e;
    int i = 0;

    if (p == 0) return;
    while (p != 0) {
        e = curnode->FindEdge(p);
        if (e == NULL) return;
        e->AdjustEdge(occurrence);
        curnode = e->RightNode();
        p = payload[++i];
    }

    e = curnode->FindEdge('\xff');
    e->AdjustEdge(occurrence);
    return;
}
```

8.15.3.2 ConstructMatrix()

`void Markov::API::ModelMatrix::ConstructMatrix ( )`

Construct the related Matrix data for the model.

This operation can be used after importing/training to allocate and populate the matrix content.

this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

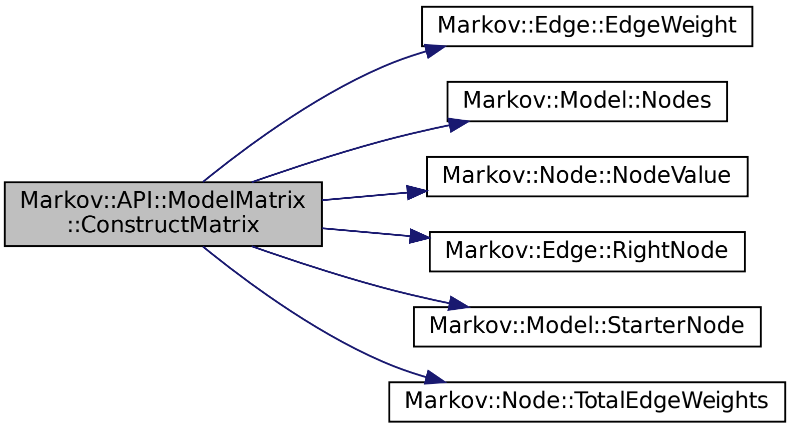
Definition at line 11 of file modelMatrix.cpp.

```
                                       {
    this->matrixSize = this->StarterNode()->edgesV.size() + 2;

    this->matrixIndex = new char[this->matrixSize];
    this->totalEdgeWeights = new long int[this->matrixSize];

    this->edgeMatrix = new char*[this->matrixSize];
    for(int i=0;i<this->matrixSize;i++){
        this->edgeMatrix[i] = new char[this->matrixSize];
    }
    this->valueMatrix = new long int*[this->matrixSize];
    for(int i=0;i<this->matrixSize;i++){
        this->valueMatrix[i] = new long int[this->matrixSize];
    }
    std::map< char, Node< char > * > *nodes;
    nodes = this->Nodes();
    int i=0;
    for (auto const& [repr, node] : *nodes){
        if(repr!=0) this->matrixIndex[i] = repr;
        else this->matrixIndex[i] = 199;
        this->totalEdgeWeights[i] = node->TotalEdgeWeights();
        for(int j=0;j<this->matrixSize;j++){
            char val = node->NodeValue();
            if(val < 0){
                for(int k=0;k<this->matrixSize;k++){
                    this->valueMatrix[i][k] = 0;
                    this->edgeMatrix[i][k] = 255;
                }
                break;
            }
            else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
                this->valueMatrix[i][j] = 0;
                this->edgeMatrix[i][j] = 255;
            }else if(j==(this->matrixSize-1)) {
                this->valueMatrix[i][j] = 0;
                this->edgeMatrix[i][j] = 255;
            }else{
                this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
                this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
            }

        }
        i++;
    }

    //this->DumpJSON();
}
```

References edgeMatrix, Markov::Edge< NodeStorageType >::EdgeWeight(), matrixIndex, matrixSize, Markov::Model< NodeStorage
Markov::Node< storageType >::NodeValue(), Markov::Edge< NodeStorageType >::RightNode(), Markov::Model< NodeStorageTyp
totalEdgeWeights, Markov::Node< storageType >::TotalEdgeWeights(), and valueMatrix.

Referenced by main().

Here is the call graph for this function:


Markov::API::ModelMatrix ::ConstructMatrix
Markov::Edge::EdgeWeight
Markov::Model::Nodes
Markov::Node::NodeValue
Markov::Edge::RightNode
Markov::Model::StarterNode
Markov::Node::TotalEdgeWeights

Here is the caller graph for this function:

main
Markov::API::ModelMatrix ::ConstructMatrix

8.15.3.3 DumpJSON()

`void Markov::API::ModelMatrix::DumpJSON ( )`

Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.

Definition at line 60 of file modelMatrix.cpp.

```
                                    {

        std::cout « "{\n    \"index\": \"";
        for(int i=0;i<this->matrixSize;i++){
            if(this->matrixIndex[i]=='"') std::cout « "\\\"";
            else if(this->matrixIndex[i]=='\\') std::cout « "\\\\";
            else if(this->matrixIndex[i]==0) std::cout « "\\\\x00";
            else if(i==0) std::cout « "\\\\xff";
            else if(this->matrixIndex[i]=='\n') std::cout « "\\n";
            else std::cout « this->matrixIndex[i];
```

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

References edgeMatrix, matrixIndex, matrixSize, and valueMatrix.

8.15.3.4 Edges()

`std::vector<Edge<char >*>* Markov::Model< char >::Edges ( )  [inline], [inherited]`

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 172 of file model.h.

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

8.15.3.5 Export() [1/2]

```
bool Markov::Model< char >::Export (
            const char * filename )  [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

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

Definition at line 285 of file model.h.

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

8.15.3.6 Export() [2/2]

```
bool Markov::Model< char >::Export (
            std::ofstream * f ) [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

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

Definition at line 273 of file model.h.

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

8.15.3.7 FastRandomWalk()

```
void Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )
```

Random walk on the Matrix-reduced Markov::Model.

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5

This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 163 of file modelMatrix.cpp.

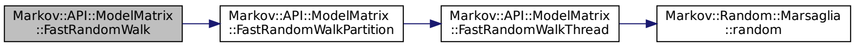
```
00163 
                                            {
00164 
```

```

    std::ofstream wordlist;
    if(bFileIO)
        wordlist.open(wordlistFileName);

    std::mutex mlock;
    if(n<=50000000ull) return this->FastRandomWalkPartition(&mlock, &wordlist, n, minLen, maxLen,
 bFileIO, threads);
    else{
        int numberOfPartitions = n/50000000ull;
        for(int i=0;i<numberOfPartitions;i++)
            this->FastRandomWalkPartition(&mlock, &wordlist, 50000000ull, minLen, maxLen, bFileIO,
 threads);
    }


}
```

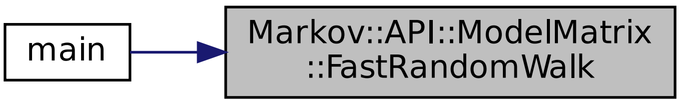
References FastRandomWalkPartition().

Referenced by main().

Here is the call graph for this function:


Markov::API::ModelMatrix ::FastRandomWalk
Markov::API::ModelMatrix ::FastRandomWalkPartition
Markov::API::ModelMatrix ::FastRandomWalkThread
Markov::Random::Marsaglia ::random

Here is the caller graph for this function:



8.15.3.8 FastRandomWalkPartition()

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

A single partition of FastRandomWalk event.

Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation
- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

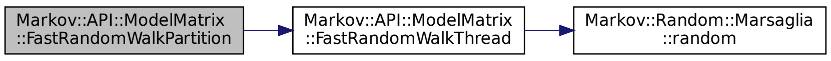
Parameters

mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

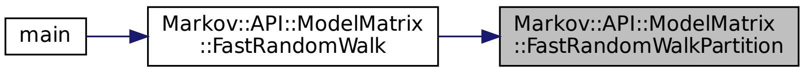
Definition at line 182 of file modelMatrix.cpp.

```
                                                                  {

    int iterationsPerThread = n/threads;
    int iterationsPerThreadCarryOver = n%threads;

    std::vector<std::thread*> threadsV;

    int id = 0;
    for(int i=0;i<threads;i++){
        threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
  mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
        id++;
    }

    threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
  wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));

    for(int i=0;i<threads;i++){
        threadsV[i]->join();
    }
}
```

References FastRandomWalkThread().
Referenced by FastRandomWalk().
Here is the call graph for this function:


Markov::API::ModelMatrix ::FastRandomWalkPartition
Markov::API::ModelMatrix ::FastRandomWalkThread
Markov::Random::Marsaglia ::random

Here is the caller graph for this function:


main
Markov::API::ModelMatrix ::FastRandomWalk
Markov::API::ModelMatrix ::FastRandomWalkPartition

8.15.3.9 FastRandomWalkThread()

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

A single thread of a single partition of FastRandomWalk.
A FastRandomWalkPartition will initiate as many of this function as requested.
This function contains the bulk of the generation algorithm.

Parameters

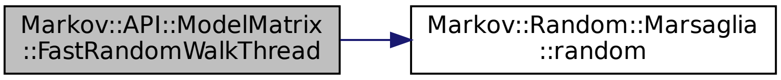
mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
id	- **DEPRECATED** Thread id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

Definition at line 112 of file modelMatrix.cpp.

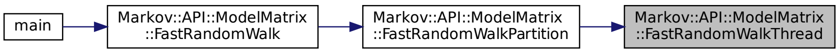
```
00112
                                                                {
00113     if(n==0) return;
00114
00115     Markov::Random::Marsaglia MarsagliaRandomEngine;
00116     char* e;
00117     char *res = new char[maxLen*n];
00118     int index = 0;
00119     char next;
00120     int len=0;
00121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

References edgeMatrix, matrixIndex, matrixSize, Markov::Random::Marsaglia::random(), totalEdgeWeights, and valueMatrix.
Referenced by FastRandomWalkPartition().

Here is the call graph for this function:


Markov::API::ModelMatrix ::FastRandomWalkThread
Markov::Random::Marsaglia ::random

Here is the caller graph for this function:


main
Markov::API::ModelMatrix ::FastRandomWalk
Markov::API::ModelMatrix ::FastRandomWalkPartition
Markov::API::ModelMatrix ::FastRandomWalkThread

8.15.3.10 Generate()

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

Call Markov::Model::RandomWalk n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

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

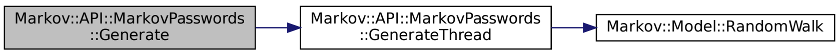
Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

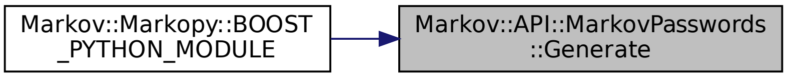
Definition at line 92 of file markovPasswords.cpp.

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

    for(int i=0;i<threads;i++){
        threadsV[i]->join();
        delete threadsV[i];
    }

    this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);

}
```

References Markov::API::MarkovPasswords::GenerateThread().
Referenced by Markov::Markopy::BOOST_PYTHON_MODULE().
Here is the call graph for this function:


Markov::API::MarkovPasswords ::Generate
Markov::API::MarkovPasswords ::GenerateThread
Markov::Model::RandomWalk

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::API::MarkovPasswords ::Generate

8.15.3.11 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
             std::mutex * outputLock,
             unsigned long int n,
             std::ofstream * wordlist,
             int minLen,
             int maxLen )  [private], [inherited]
```

A single thread invoked by the Generate function.

DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

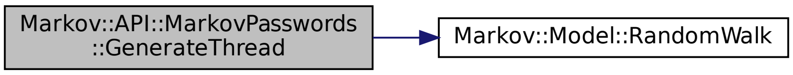
Parameters

outputLock	- shared mutex lock to lock during output operation. Prevents race condition on write.
n	number of lines to be generated by this thread
wordlist	wordlistfile
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

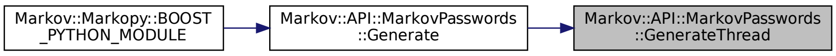
Definition at line 114 of file markovPasswords.cpp.

```

                                      {
    char* res = new char[maxLen+5];
    if(n==0) return;

    Markov::Random::Marsaglia MarsagliaRandomEngine;
    for (int i = 0; i < n; i++) {
        this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
        outputLock->lock();
        *wordlist « res « "\n";
        outputLock->unlock();
    }
}
```

References Markov::Model< NodeStorageType >::RandomWalk().
Referenced by Markov::API::MarkovPasswords::Generate().
Here is the call graph for this function:


Markov::API::MarkovPasswords ::GenerateThread
Markov::Model::RandomWalk

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::API::MarkovPasswords ::Generate
Markov::API::MarkovPasswords ::GenerateThread

8.15.3.12 Import() [1/2]

```
bool Markov::Model< char >::Import (
             const char * filename )  [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

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

Definition at line 265 of file model.h.

```
                                                            {
    std::ifstream importfile;
    importfile.open(filename);
    return this->Import(&importfile);

}
```

8.15.3.13 Import() [2/2]

```
bool Markov::Model< char >::Import (
             std::ifstream * f )  [inherited]
```

Import a file to construct the model.
File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr
Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

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

Definition at line 206 of file model.h.

```
                                                      {
    std::string cell;

    char src;
    char target;
    long int oc;

    while (std::getline(*f, cell)) {
        //std::cout « "cell: " « cell « std::endl;
        src = cell[0];
        target = cell[cell.length() - 1];
        char* j;
        oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
        //std::cout « oc « "\n";
        Markov::Node<NodeStorageType>* srcN;
        Markov::Node<NodeStorageType>* targetN;
        Markov::Edge<NodeStorageType>* e;
        if (this->nodes.find(src) == this->nodes.end()) {
            srcN = new Markov::Node<NodeStorageType>(src);
            this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
            //std::cout « "Creating new node at start.\n";
        }
        else {
            srcN = this->nodes.find(src)->second;
        }

        if (this->nodes.find(target) == this->nodes.end()) {
            targetN = new Markov::Node<NodeStorageType>(target);
            this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
            //std::cout « "Creating new node at end.\n";
        }
        else {
            targetN = this->nodes.find(target)->second;
        }
        e = srcN->Link(targetN);
        e->AdjustEdge(oc);
        this->edges.push_back(e);

        //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " « int(targetN->NodeValue()) « "\n";


    }

    for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
        //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
        std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs, Edge<NodeStorageType> *rhs)->bool{
            return lhs->EdgeWeight() > rhs->EdgeWeight();
        });
        //for(int i=0;i<x.second->edgesV.size();i++)
        //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
        //std::cout « "\n";
    }
    //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
    //std::cout « "Total number of edges: " « this->edges.size() « std::endl;

    return true;
}
```

8.15.3.14 Nodes()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 177 of file model.h.

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

8.15.3.15 OpenDatasetFile()

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

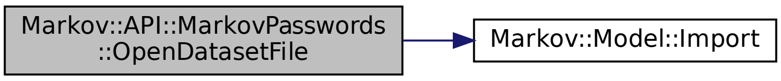
ifstream∗ to the the dataset file

Definition at line 27 of file markovPasswords.cpp.

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

References Markov::Model< NodeStorageType >::Import().

Here is the call graph for this function:



8.15.3.16 RandomWalk()

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 292 of file model.h.

```
                                                                                    {
    Markov::Node<NodeStorageType>* n = this->starterNode;
    int len = 0;
    Markov::Node<NodeStorageType>* 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();
    }

    //null terminate the string
    buffer[len] = 0x00;

    //do something with the generated string
    return buffer; //for now
}
```

8.15.3.17 Save()

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

Export model to file.

Parameters

filename	- Export filename.

Returns

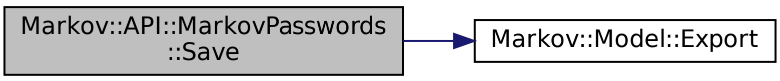
std::ofstream* of the exported file.

Definition at line 80 of file markovPasswords.cpp.

```
                                                                                    {
    std::ofstream* exportFile;

    std::ofstream newFile(filename);

    exportFile = &newFile;

    this->Export(exportFile);
    return exportFile;
}
```

References Markov::Model< NodeStorageType >::Export().

Here is the call graph for this function:



8.15.3.18 StarterNode()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 167 of file model.h.

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

8.15.3.19 Train()

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

Train the model with the dataset file.

Parameters

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

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

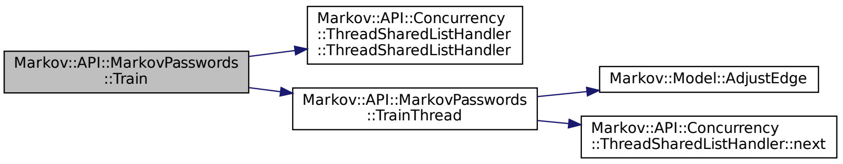
Definition at line 40 of file markovPasswords.cpp.

```
00040                                                                                           {
00041     Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
00042     auto start = std::chrono::high_resolution_clock::now();
00043 
00044     std::vector<std::thread*> threadsV;
00045     for(int i=0;i<threads;i++){
00046         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
       &listhandler, delimiter));
00047     }
00048 
00049     for(int i=0;i<threads;i++){
00050         threadsV[i]->join();
00051         delete threadsV[i];
00052     }
00053     auto finish = std::chrono::high_resolution_clock::now();
00054     std::chrono::duration<double> elapsed = finish - start;
00055     std::cout « "Elapsed time: " « elapsed.count() « " s\n";
00056 
00057 }
```

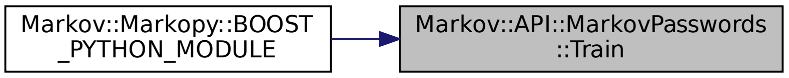
References Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler(), and Markov::API::MarkovPasswords::

Referenced by Markov::Markopy::BOOST_PYTHON_MODULE().

Here is the call graph for this function:


Markov::API::MarkovPasswords ::Train
Markov::API::Concurrency ::ThreadSharedListHandler ::ThreadSharedListHandler
Markov::API::MarkovPasswords ::TrainThread
Markov::Model::AdjustEdge
Markov::API::Concurrency ::ThreadSharedListHandler::next

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::API::MarkovPasswords ::Train

8.15.3.20 TrainThread()

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

A single thread invoked by the Train function.

Parameters

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

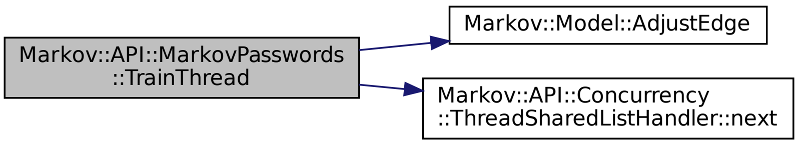
Definition at line 59 of file markovPasswords.cpp.

```
                  {
        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];
    #ifdef _WIN32
            sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
    #else
            sscanf(line.c_str(), format_str, &oc, linebuf);
    #endif
            this->AdjustEdge((const char*)linebuf, oc);
            delete linebuf;
        }
    }
```

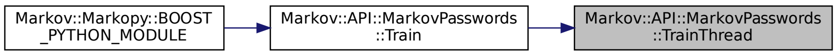
References Markov::Model< NodeStorageType >::AdjustEdge(), and Markov::API::Concurrency::ThreadSharedListHandler::next().

Referenced by Markov::API::MarkovPasswords::Train().

Here is the call graph for this function:


Markov::API::MarkovPasswords ::TrainThread
Markov::Model::AdjustEdge
Markov::API::Concurrency ::ThreadSharedListHandler::next

Here is the caller graph for this function:


Markov::Markopy::BOOST _PYTHON_MODULE
Markov::API::MarkovPasswords ::Train
Markov::API::MarkovPasswords ::TrainThread

8.15.4 Member Data Documentation

8.15.4.1 datasetFile

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

8.15.4.2 edgeMatrix

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

8.15.4.3 edges

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

8.15.4.4 matrixIndex

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

8.15.4.5 matrixSize

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

8.15.4.6 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]`
Definition at line 107 of file markovPasswords.h.

8.15.4.7 nodes

`std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]`
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 183 of file model.h.

8.15.4.8 outputFile

`std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]`
Definition at line 108 of file markovPasswords.h.

8.15.4.9 starterNode

`Node<char >* Markov::Model< char >::starterNode [private], [inherited]`
Starter Node of this model.
Definition at line 188 of file model.h.

8.15.4.10 totalEdgeWeights

`long int* Markov::API::ModelMatrix::totalEdgeWeights [protected]`
Definition at line 116 of file modelMatrix.h.
Referenced by ConstructMatrix(), and FastRandomWalkThread().

8.15.4.11 valueMatrix

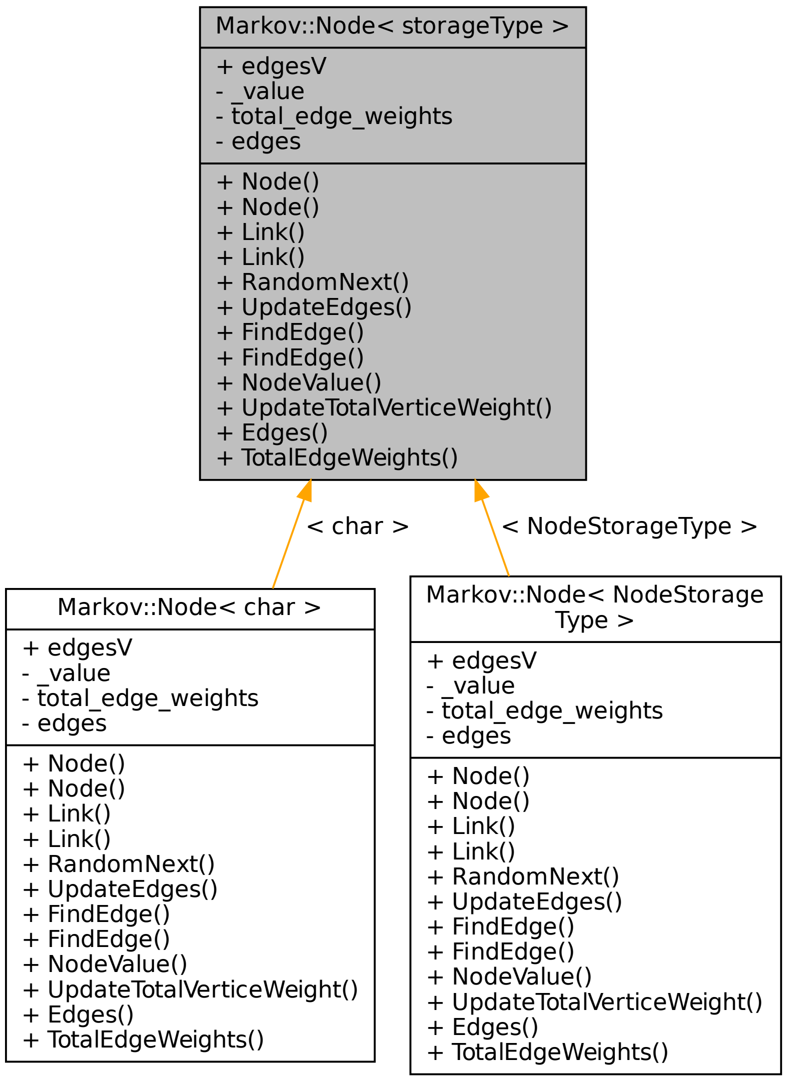
`long int** Markov::API::ModelMatrix::valueMatrix [protected]`
Definition at line 113 of file modelMatrix.h.
Referenced by ConstructMatrix(), DumpJSON(), and FastRandomWalkThread().
The documentation for this class was generated from the following files:

- modelMatrix.h
- modelMatrix.cpp

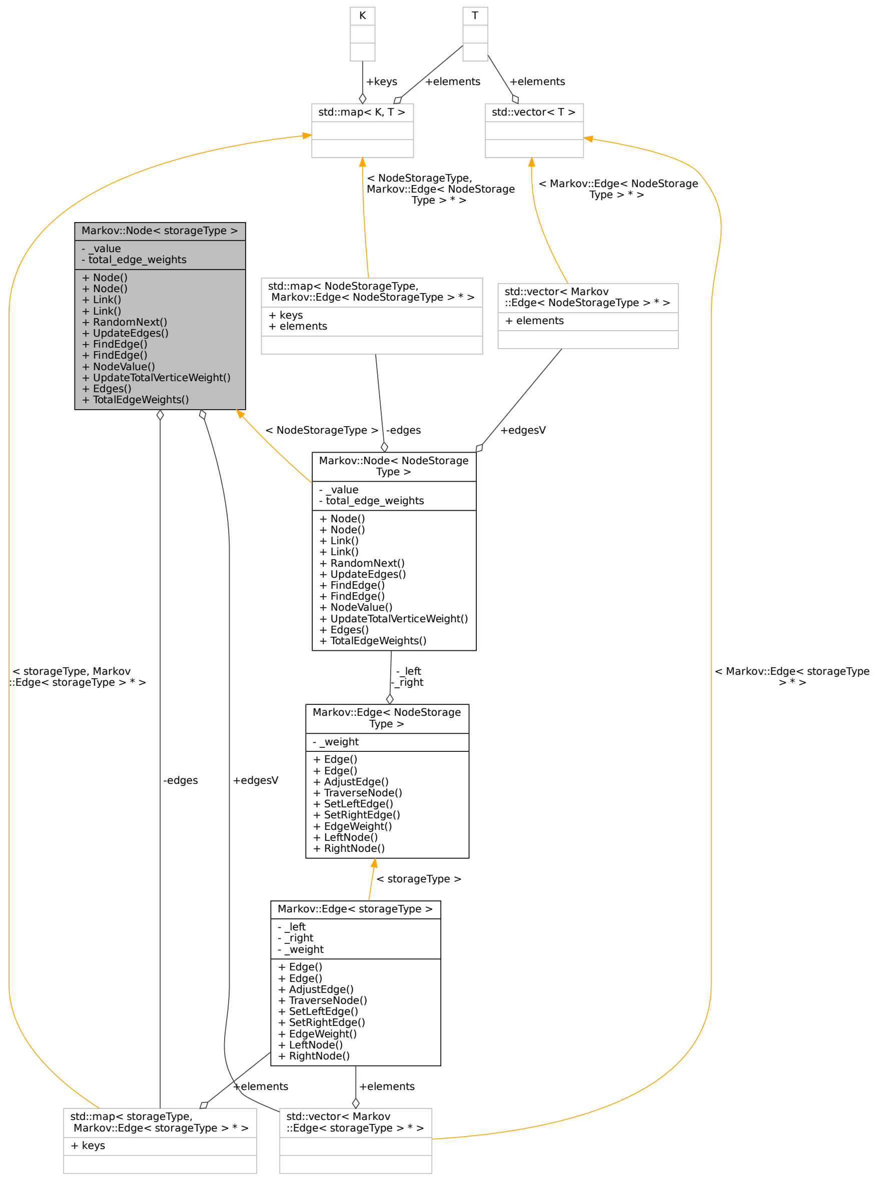
8.16 Markov::Node< storageType > Class Template Reference

A node class that for the vertices of model. Connected with eachother using Edge.
`#include <model.h>`

Inheritance diagram for Markov::Node< storageType >:



Collaboration diagram for Markov::Node< storageType >:



Public Member Functions

- Node ()

  Default constructor. Creates an empty Node.
- Node (storageType _value)

  Constructor. Creates a Node with no edges and with given NodeValue.
- Edge< storageType > ∗ Link (Node< storageType > ∗)

  Link this node with another, with this node as its source.
- Edge< storageType > ∗ Link (Edge< storageType > ∗)

  Link this node with another, with this node as its source.
- Node< storageType > ∗ RandomNext (Markov::Random::RandomEngine ∗randomEngine)

  Chose a random node from the list of edges, with regards to its EdgeWeight, and TraverseNode to that.
- bool UpdateEdges (Edge< storageType > ∗)

  Insert a new edge to the this.edges.

- Edge< storageType > ∗ FindEdge (storageType repr)

  Find an edge with its character representation.
- Edge< storageType > ∗ FindEdge (Node< storageType > ∗target)

  Find an edge with its pointer. Avoid unless neccessary because comptutational cost of find by character is cheaper (because of std::map)
- unsigned char NodeValue ()

  Return character representation of this node.
- void UpdateTotalVerticeWeight (long int offset)

  Change total weights with offset.
- std::map< storageType, Edge< storageType > ∗ > ∗ Edges ()

  return edges
- long int TotalEdgeWeights ()

  return total edge weights

Public Attributes

- std::vector< Edge< storageType > ∗ > edgesV

Private Attributes

- storageType _value
- long int total_edge_weights

  Character representation of this node. 0 for starter, 0xff for terminator.
- std::map< storageType, Edge< storageType > ∗ > edges

  Total weights of the vertices, required by RandomNext;.

8.16.1 Detailed Description

template<typename storageType>
class Markov::Node< storageType >

A node class that for the vertices of model. Connected with eachother using Edge.
This class will later be templated to accept other data types than char∗.
Definition at line 23 of file model.h.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 Node() [1/2]

```
template<typename storageType >
Markov::Node< storageType >::Node
```

Default constructor. Creates an empty Node.
Definition at line 196 of file node.h.

```
00196                           {
00197     this->_value = 0;
00198     this->total_edge_weights = 0L;
00199 };
```

8.16.2.2 Node() [2/2]

```
template<typename storageType >
Markov::Node< storageType >::Node (
                storageType _value )
```

Constructor. Creates a Node with no edges and with given NodeValue.

Parameters

_value	- 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.

```
                                                    {
    this->_value = _value;
    this->total_edge_weights = 0L;
};
```

8.16.3 Member Function Documentation

8.16.3.1 Edges()

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

return edges

Definition at line 259 of file node.h.

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

8.16.3.2 FindEdge() [1/2]

```
template<typename storageType >
Edge<storageType>* Markov::Node< storageType >::FindEdge (
            Node< storageType > * target )
```

Find an edge with its pointer. Avoid unless neccessary because comptutational cost of find by character is cheaper (because of std::map)

Parameters

target	- target node.

Returns

Edge that is connected between this node, and the target node.

8.16.3.3 FindEdge() [2/2]

```
template<typename storageType >
Markov::Edge< storageType > * Markov::Node< storageType >::FindEdge (
            storageType repr )
```

Find an edge with its character representation.

Parameters

repr	- character NodeValue of the target node.

Returns

Edge that is connected between this node, and the target node.

Example Use: Construct and update edges

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

Definition at line 247 of file node.h.

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

8.16.3.4 Link() [1/2]

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

Link this node with another, with this node as its source.
DOES NOT create a new Edge.

Parameters

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

Returns

the same edge as parameter target.

Example Use: Construct and link nodes

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

Definition at line 214 of file node.h.

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

8.16.3.5 Link() [2/2]

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

Link this node with another, with this node as its source.
Creates a new Edge.

Parameters

target	- 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.

```
                                                                              {
    Markov::Edge<storageType>* v = new Markov::Edge<storageType>(this, n);
    this->UpdateEdges(v);
    return v;
}
```

8.16.3.6 NodeValue()

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

Return character representation of this node.

Returns

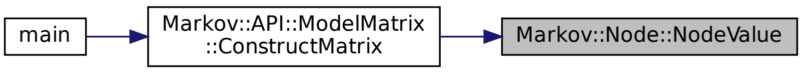
character representation at _value.

Definition at line 202 of file node.h.

```
                                                  {
    return _value;
}
```

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

Here is the caller graph for this function:



8.16.3.7 RandomNext()

template<typename storageType >
Markov::Node< storageType > * Markov::Node< storageType >::RandomNext (
Markov::Random::RandomEngine * *randomEngine*)

Chose a random node from the list of edges, with regards to its EdgeWeight, and TraverseNode to that.

This operation is done by generating a random number in range of 0-this.total_edge_weights, and then iterating over the list of edges. At each step, EdgeWeight of the edge is subtracted from the random number, and once it is 0, next node is selected.

Returns

Node that was chosen at EdgeWeight biased random.

Example Use: Use randomNext to do a random walk on the model

```
char* buffer[64];
Markov::Model<char> model;
model.Import("model.mdl");
   Markov::Node<char>* n = model.starterNode;
int len = 0;
Markov::Node<char>* temp_node;
while (true) {
   temp_node = n->RandomNext(randomEngine);
   if (len >= maxSetting) {
      break;
   }
   else if ((temp_node == NULL) && (len < minSetting)) {
      continue;
   }
   else if (temp_node == NULL){
      break;
   }
```

```
    n = temp_node;
    buffer[len++] = n->NodeValue();
}
```

Definition at line 221 of file node.h.

```
                                                                                              {

    //get a random NodeValue in range of total_vertice_weight
    long int selection = randomEngine->random() %
 this->total_edge_weights;//distribution()(generator());// distribution(generator);
    //make absolute, no negative modulus values wanted
    //selection = (selection >= 0) ? selection : (selection + this->total_edge_weights);
    for(int i=0;i<this->edgesV.size();i++){
        selection -= this->edgesV[i]->EdgeWeight();
        if (selection < 0) return this->edgesV[i]->TraverseNode();
    }

    //if this assertion is reached, it means there is an implementation error above
    std::cout « "This should never be reached (node failed to walk to next)\n"; //cant assert from
 child thread
    assert(true && "This should never be reached (node failed to walk to next)");
    return NULL;
}
```

8.16.3.8 TotalEdgeWeights()

```
template<typename storageType >
long int Markov::Node< storageType >::TotalEdgeWeights [inline]
```

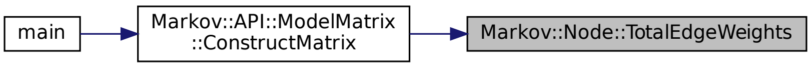
return total edge weights

Definition at line 264 of file node.h.

```
                                                            {
    return this->total_edge_weights;
}
```

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

Here is the caller graph for this function:


main
Markov::API::ModelMatrix ::ConstructMatrix
Markov::Node::TotalEdgeWeights

8.16.3.9 UpdateEdges()

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

Insert a new edge to the this.edges.

Parameters

edge	- 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.

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

8.16.3.10 UpdateTotalVerticeWeight()

```
template<typename storageType >
void Markov::Node< storageType >::UpdateTotalVerticeWeight (
            long int offset )
```

Change total weights with offset.

Parameters

offset	to adjust the vertice weight with

Definition at line 254 of file node.h.

```
                                                                    {
    this->total_edge_weights += offset;
}
```

8.16.4 Member Data Documentation

8.16.4.1 _value

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

Definition at line 169 of file node.h.

Referenced by Markov::Node< NodeStorageType >::NodeValue().

8.16.4.2 edges

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

Total weights of the vertices, required by RandomNext;.

A map of all edges connected to this node, where this node is at the LeftNode.

Map is indexed by unsigned char, which is the character representation of the node.

Definition at line 177 of file node.h.

8.16.4.3 edgesV

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

Definition at line 165 of file node.h.

8.16.4.4 total_edge_weights

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

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

Definition at line 171 of file node.h.

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

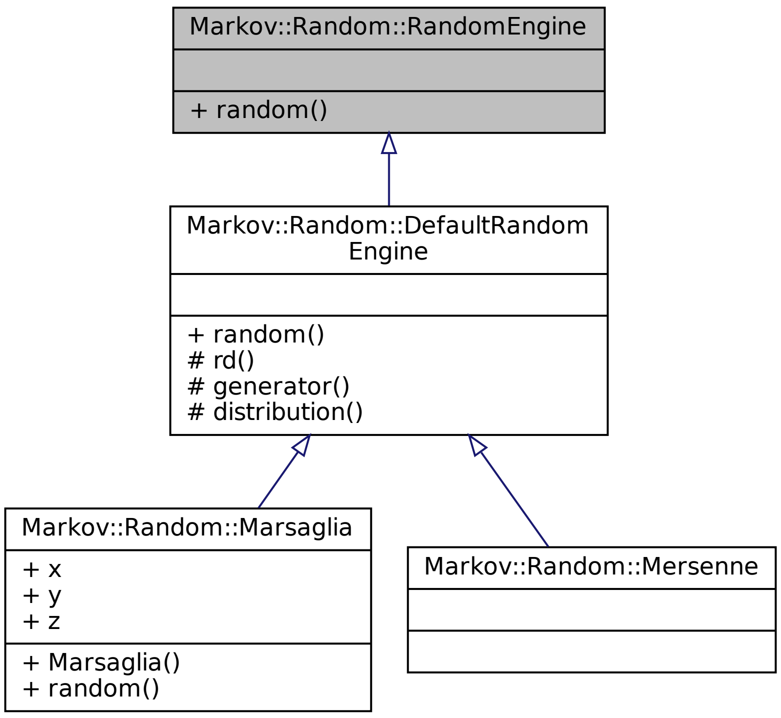
- model.h
- node.h

8.17 Markov::Random::RandomEngine Class Reference

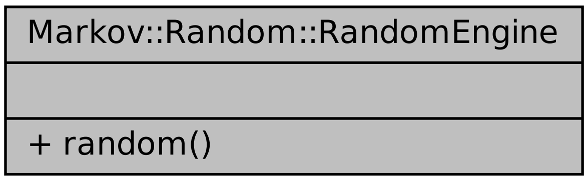
An abstract class for Random Engine.

`#include <random.h>`

Inheritance diagram for Markov::Random::RandomEngine:



Collaboration diagram for Markov::Random::RandomEngine:



Public Member Functions

- virtual unsigned long random ()=0

8.17.1 Detailed Description

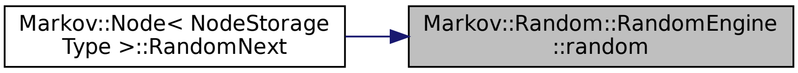
An abstract class for Random Engine.
This class is used for generating random numbers, which are used for random walking on the graph.
Main reason behind allowing different random engines is that some use cases may favor performance, while some favor good random.
Mersenne can be used for truer random, while Marsaglia can be used for deterministic but fast random.
Definition at line 21 of file random.h.

8.17.2 Member Function Documentation

8.17.2.1 random()

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

Implemented in Markov::Random::Marsaglia, and Markov::Random::DefaultRandomEngine.
Referenced by Markov::Node< NodeStorageType >::RandomNext().
Here is the caller graph for this function:


Markov::Node< NodeStorage Type >::RandomNext
Markov::Random::RandomEngine ::random

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

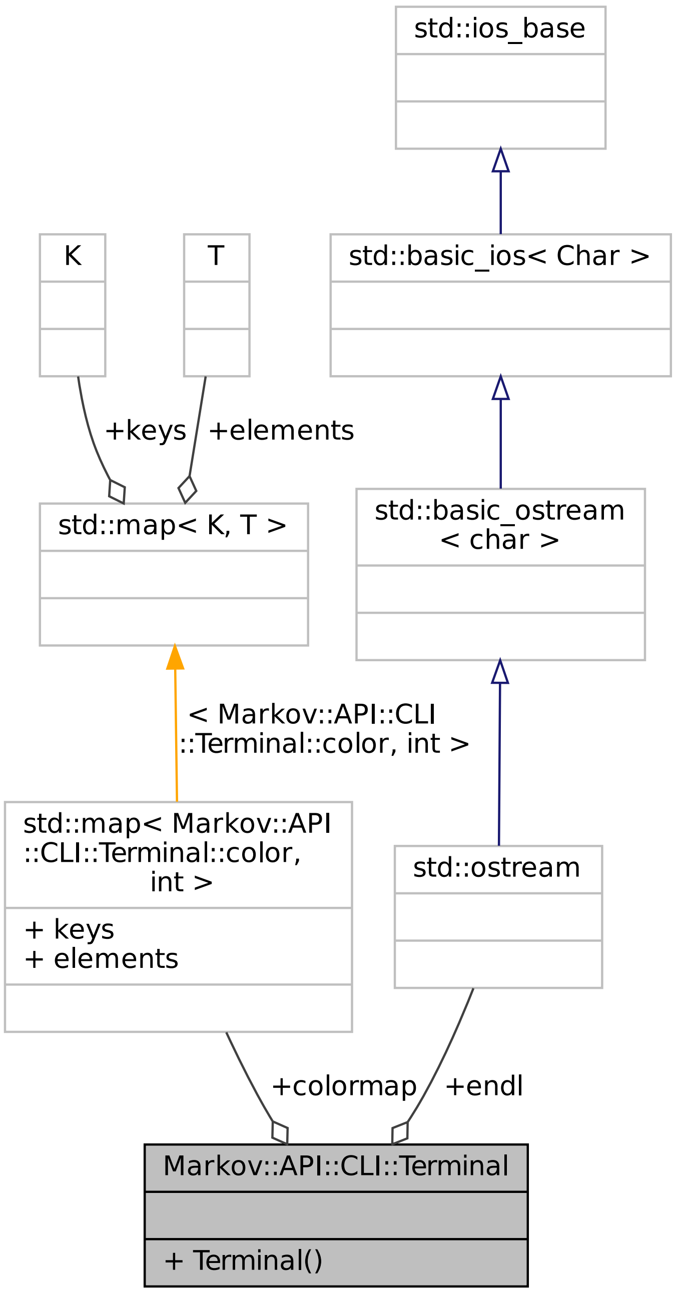
- random.h

8.18 Markov::API::CLI::Terminal Class Reference

pretty colors for Terminal. Windows Only.

```
#include <term.h>
```

Collaboration diagram for Markov::API::CLI::Terminal:


std::ios_base
K
T
std::basic_ios< Char >
+keys
+elements
std::map< K, T >
std::basic_ostream < char >
< Markov::API::CLI ::Terminal::color, int >
std::map< Markov::API ::CLI::Terminal::color, int >
+ keys
+ elements
std::ostream
+colormap
+endl
Markov::API::CLI::Terminal
+ Terminal()

Public Types

- enum color {
  RESET, BLACK, RED, GREEN,
  YELLOW, BLUE, MAGENTA, CYAN,
  WHITE, LIGHTGRAY, DARKGRAY, BROWN }

Public Member Functions

- Terminal ()

Static Public Attributes

- static std::map< Markov::API::CLI::Terminal::color, int > colormap
- static std::ostream endl

8.18.1 Detailed Description

pretty colors for Terminal. Windows Only.
Definition at line 18 of file term.h.

8.18.2 Member Enumeration Documentation

8.18.2.1 color

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

Enumerator

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

Definition at line 26 of file term.h.

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

8.18.3 Constructor & Destructor Documentation

8.18.3.1 Terminal()

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

Default constructor. Get references to stdout and stderr handles.
Definition at line 56 of file term.cpp.

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

8.18.4 Member Data Documentation

8.18.4.1 colormap

```
std::map< Terminal::color, int > Terminal::colormap  [static]
```

Initial value:

```
= {
    {Terminal::color::BLACK, 30},
    {Terminal::color::BLUE, 34},
    {Terminal::color::GREEN, 32},
    {Terminal::color::CYAN, 36},
    {Terminal::color::RED, 31},
    {Terminal::color::MAGENTA, 35},
    {Terminal::color::BROWN, 0},
    {Terminal::color::LIGHTGRAY, 0},
    {Terminal::color::DARKGRAY, 0},
    {Terminal::color::YELLOW, 33},
    {Terminal::color::WHITE, 37},
    {Terminal::color::RESET, 0},
}
```

Definition at line 32 of file term.h.
Referenced by operator<<().

8.18.4.2 endl

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

Definition at line 37 of file term.h.
The documentation for this class was generated from the following files:

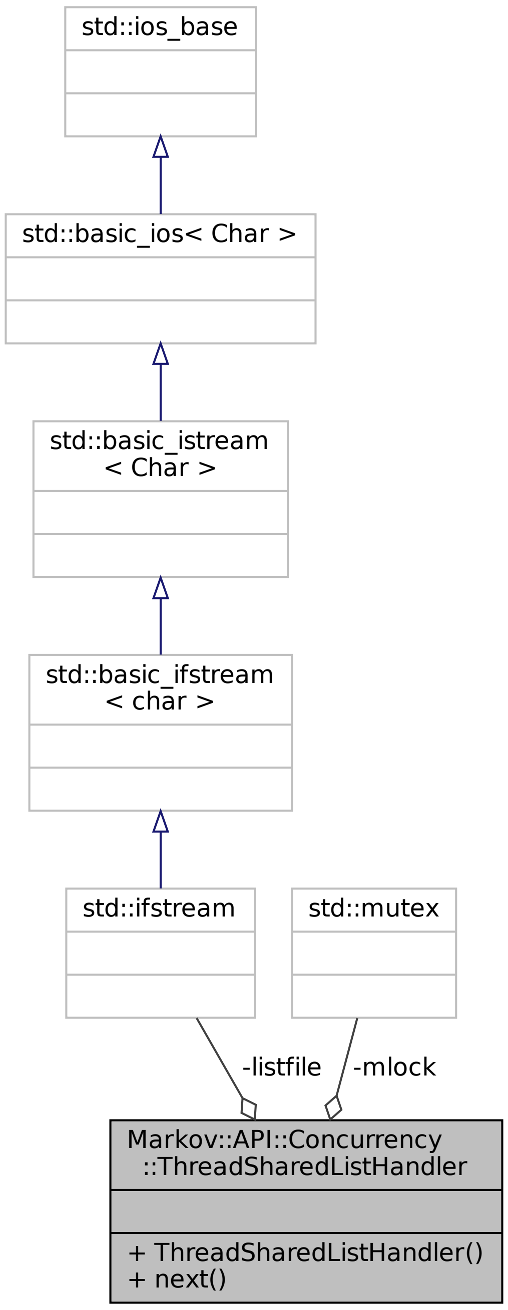
- term.h
- term.cpp

8.19 Markov::API::Concurrency::ThreadSharedListHandler Class Reference

Simple class for managing shared access to file.

```
#include <threadSharedListHandler.h>
```

Collaboration diagram for Markov::API::Concurrency::ThreadSharedListHandler:


std::ios_base
std::basic_ios< Char >
std::basic_istream < Char >
std::basic_ifstream < char >
std::ifstream
std::mutex
-listfile
-mlock
Markov::API::Concurrency ::ThreadSharedListHandler
+ ThreadSharedListHandler()
+ next()

Public Member Functions

- ThreadSharedListHandler (const char ∗filename)

  Construct the Thread Handler with a filename.

- bool next (std::string ∗line)

  Read the next line from the file.

Private Attributes

- std::ifstream listfile
- std::mutex mlock

8.19.1 Detailed Description

Simple class for managing shared access to file.
This class maintains the handover of each line from a file to multiple threads.
When two different threads try to read from the same file while reading a line isn't completed, it can have unexpected results. Line might be split, or might be read twice. This class locks the read action on the list until a line is completed, and then proceeds with the handover.
Definition at line 18 of file threadSharedListHandler.h.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 ThreadSharedListHandler()

```
Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler (
                const char * filename )
```

Construct the Thread Handler with a filename.
Simply open the file, and initialize the locks.
Example Use: Simple file read

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

Example Use: Example use case from MarkovPasswords showing multithreaded access

```
void MarkovPasswords::Train(const char* datasetFileName, char delimiter, int threads)   {
     ThreadSharedListHandler listhandler(datasetFileName);
     auto start = std::chrono::high_resolution_clock::now();
     std::vector<std::thread*> threadsV;
     for(int i=0;i<threads;i++){
         threadsV.push_back(new std::thread(&MarkovPasswords::TrainThread, this, &listhandler,
       datasetFileName, delimiter));
     }
    for(int i=0;i<threads;i++){
         threadsV[i]->join();
         delete threadsV[i];
     }
     auto finish = std::chrono::high_resolution_clock::now();
     std::chrono::duration<double> elapsed = finish - start;
     std::cout « "Elapsed time: " « elapsed.count() « " s\n";
 }
 void MarkovPasswords::TrainThread(ThreadSharedListHandler *listhandler, const char* datasetFileName, char
       delimiter){
     char format_str[] ="%ld,%s";
     format_str[2]=delimiter;
     std::string line;
     while (listhandler->next(&line)) {
         long int oc;
         if (line.size() > 100) {
             line = line.substr(0, 100);
         }
         char* linebuf = new char[line.length()+5];
         sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
         this->AdjustEdge((const char*)linebuf, oc);
         delete linebuf;
     }
 }
```

Parameters

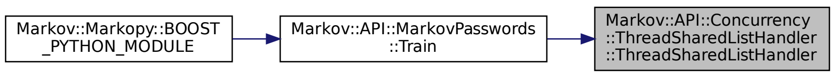
filename	Filename for the file to manage.

Definition at line 4 of file threadSharedListHandler.cpp.

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

References listfile.

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


Markov::Markopy::BOOST_PYTHON_MODULE
Markov::API::MarkovPasswords::Train
Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler

8.19.3 Member Function Documentation

8.19.3.1 next()

```
bool Markov::API::Concurrency::ThreadSharedListHandler::next (
                std::string * line )
```

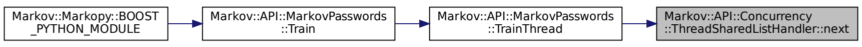
Read the next line from the file.
This action will be blocked until another thread (if any) completes the read operation on the file.
Example Use: Simple file read

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

Definition at line 10 of file threadSharedListHandler.cpp.

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

References listfile, and mlock.
Referenced by Markov::API::MarkovPasswords::TrainThread().
Here is the caller graph for this function:


Markov::Markopy::BOOST_PYTHON_MODULE
Markov::API::MarkovPasswords::Train
Markov::API::MarkovPasswords::TrainThread
Markov::API::Concurrency::ThreadSharedListHandler::next

8.19.4 Member Data Documentation

8.19.4.1 listfile

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

Definition at line 88 of file threadSharedListHandler.h.
Referenced by next(), and ThreadSharedListHandler().

8.19.4.2 mlock

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

Definition at line 89 of file threadSharedListHandler.h.
Referenced by next().
The documentation for this class was generated from the following files:

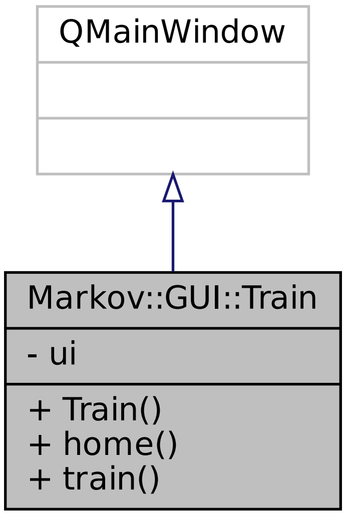
- threadSharedListHandler.h
- threadSharedListHandler.cpp

8.20 Markov::GUI::Train Class Reference

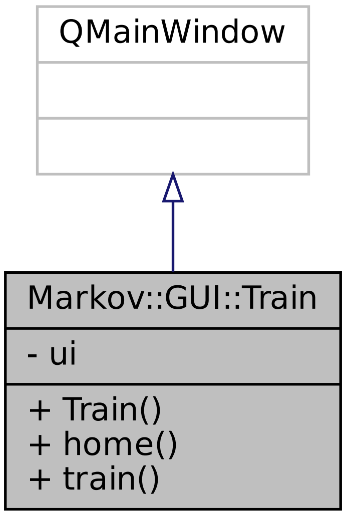
QT Training page class.

`#include <Train.h>`

Inheritance diagram for Markov::GUI::Train:



Collaboration diagram for Markov::GUI::Train:



Public Slots

- void home ()
- void train ()

Public Member Functions

- Train (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::Train ui

8.20.1 Detailed Description

QT Training page class.
Definition at line 9 of file Train.h.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 Train()

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

8.20.3 Member Function Documentation

8.20.3.1 home

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

8.20.3.2 train

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

8.20.4 Member Data Documentation

8.20.4.1 ui

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

Definition at line 15 of file Train.h.
The documentation for this class was generated from the following file:

- Train.h

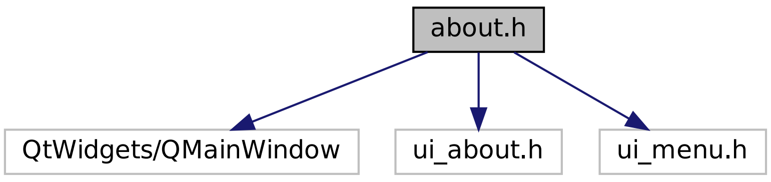
Chapter 9

File Documentation

9.1 about.h File Reference

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

Include dependency graph for about.h:



Classes

- class Markov::GUI::about

  QT Class for about page.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

- Markov::GUI

  namespace for MarkovPasswords API GUI wrapper

9.2 about.h

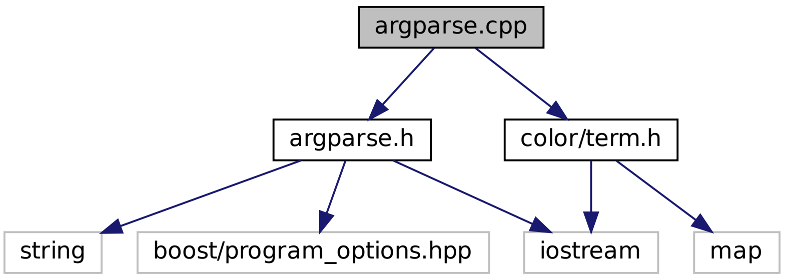
```
#pragma once
#include <QtWidgets/QMainWindow>
#include "ui_about.h"
#include <ui_menu.h>

/** @brief namespace for MarkovPasswords API GUI wrapper
*/
namespace Markov::GUI{

    /** @brief QT Class for about page
    */
    class about :public QMainWindow {
```

```
    Q_OBJECT
    public:
        about(QWidget* parent = Q_NULLPTR);

    private:
        Ui:: main ui;


    };
};
```

9.3 argparse.cpp File Reference

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

Include dependency graph for argparse.cpp:



9.4 argparse.cpp

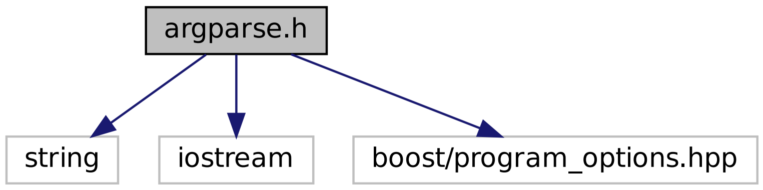
```
#include "argparse.h"
#include "color/term.h"

Markov::API::CLI::ProgramOptions* Markov::API::CLI::Argparse::parse(int argc, char** argv) { return 0;
 }



void Markov::API::CLI::Argparse::help() {
    std::cout «
    "Markov Passwords - Help\n"
    "Options:\n"
    "    \n"
    "    -of --outputfilename\n"
    "        Filename to output the generation results\n"
    "    -ef --exportfilename\n"
    "        filename to export built model to\n"
    "    -if --importfilename\n"
    "        filename to import model from\n"
    "    -n (generate count)\n"
    "        Number of lines to generate\n"
    "    \n"
    "Usage: \n"
    "    markov.exe -if empty_model.mdl -ef model.mdl\n"
    "        import empty_model.mdl and train it with data from stdin. When done, output the model to
model.mdl\n"
    "\n"
    "    markov.exe -if empty_model.mdl -n 15000 -of wordlist.txt\n"
    "        import empty_model.mdl and generate 15000 words to wordlist.txt\n"

     « std::endl;
}
```

9.5 argparse.h File Reference

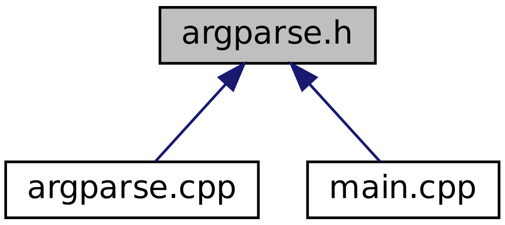
```
#include <string>
```

```
#include <iostream>
#include <boost/program_options.hpp>
```

Include dependency graph for argparse.h:



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



Classes

- struct Markov::API::CLI::_programOptions

  Structure to hold parsed cli arguements.
- class Markov::API::CLI::Argparse

  Parse command line arguements.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.
- Markov::API::CLI

  Structure to hold parsed cli arguements.

Macros

- #define BOOST_ALL_DYN_LINK 1

Typedefs

- typedef struct Markov::API::CLI::_programOptions Markov::API::CLI::ProgramOptions

  Structure to hold parsed cli arguements.

9.5.1 Macro Definition Documentation

9.5.1.1 BOOST_ALL_DYN_LINK

```
#define BOOST_ALL_DYN_LINK 1
```

Definition at line 4 of file argparse.h.

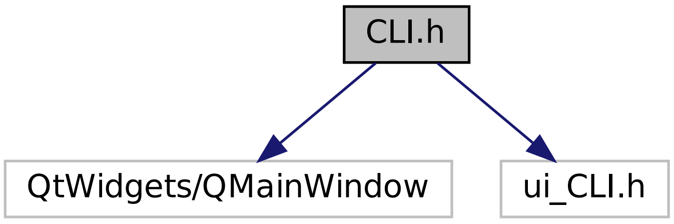
9.6 argparse.h

```
#include<string>
#include<iostream>

#define BOOST_ALL_DYN_LINK 1

#include <boost/program_options.hpp>
/** @brief Structure to hold parsed cli arguements.
*/
namespace opt = boost::program_options;

/**
    @brief Namespace for the CLI objects
*/
namespace Markov::API::CLI{

    /** @brief Structure to hold parsed cli arguements.
    */
    typedef struct _programOptions {
        bool bImport;
        bool bExport;
        bool bFailure;
        char seperator;
        std::string importname;
        std::string exportname;
        std::string wordlistname;
        std::string outputfilename;
        std::string datasetname;
        int generateN;
    } ProgramOptions;


    /** @brief Parse command line arguements.
    */
    class Argparse {
    public:

        Argparse();

        /** @brief Parse command line arguements.
         *
         * Parses command line arguements to populate ProgramOptions structure.
         *
         * @param argc Number of command line arguements
         * @param argv Array of command line parameters
        */
        Argparse(int argc, char** argv) {

            /*bool bImp;
            bool bExp;
            bool bFail;
            char sprt;
            std::string imports;
            std::string exports;
            std::string outputs;
            std::string datasets;
            int generateN;
            */
            opt::options_description desc("Options");


            desc.add_options()
                ("generate", "Generate strings with given parameters")
                ("train", "Train model with given parameters")
                ("combine", "Combine")
                ("import", opt::value<std::string>(), "Import model file")
                ("output", opt::value<std::string>(), "Output model file. This model will be exported when done. Will be ignored for generation mode")
                ("dataset", opt::value<std::string>(), "Dataset file to read input from training. Will be ignored for generation mode")
                ("seperator", opt::value<char>(), "Seperator character to use with training data. (character between occurence and value)")
                ("wordlist", opt::value<std::string>(), "Wordlist file path to export generation results to. Will be ignored for training mode")
                ("count", opt::value<int>(), "Number of lines to generate. Ignored in training mode")
                ("verbosity", "Output verbosity")
                ("help", "Option definitions");

                opt::variables_map vm;

                opt::store(opt::parse_command_line(argc, argv, desc), vm);

                opt::notify(vm);

                //std::cout « desc « std::endl;
                if (vm.count("help")) {
                std::cout « desc « std::endl;
                }

                if (vm.count("output") == 0) this->po.outputfilename = "NULL";
                else if (vm.count("output") == 1) {
                    this->po.outputfilename = vm["output"].as<std::string>();
                    this->po.bExport = true;
                }
                else {
                    this->po.bFailure = true;
                    std::cout « "UNIDENTIFIED INPUT" « std::endl;
                    std::cout « desc « std::endl;
                }


                if (vm.count("dataset") == 0) this->po.datasetname = "NULL";
                else if (vm.count("dataset") == 1) {
                    this->po.datasetname = vm["dataset"].as<std::string>();
                }
                else {
                    this->po.bFailure = true;
                    std::cout « "UNIDENTIFIED INPUT" « std::endl;
                    std::cout « desc « std::endl;
                }


                if (vm.count("wordlist") == 0) this->po.wordlistname = "NULL";
                else if (vm.count("wordlist") == 1) {
                    this->po.wordlistname = vm["wordlist"].as<std::string>();
                }
                else {
                    this->po.bFailure = true;
                    std::cout « "UNIDENTIFIED INPUT" « std::endl;
                    std::cout « desc « std::endl;
                }

                if (vm.count("import") == 0) this->po.importname = "NULL";
                else if (vm.count("import") == 1) {
                    this->po.importname = vm["import"].as<std::string>();
                    this->po.bImport = true;
                }
                else {
                    this->po.bFailure = true;
                    std::cout « "UNIDENTIFIED INPUT" « std::endl;
                    std::cout « desc « std::endl;
                }


                if (vm.count("count") == 0) this->po.generateN = 0;
                else if (vm.count("count") == 1) {
                    this->po.generateN = vm["count"].as<int>();
                }
                else {
                    this->po.bFailure = true;
                    std::cout « "UNIDENTIFIED INPUT" « std::endl;
                    std::cout « desc « std::endl;
                }

                /*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;
                std::cout « vm["count"].as<int>() « std::endl;*/


                //else if (vm.count("train")) std::cout « "train oldu" « std::endl;
        }

        /** @brief Getter for command line options
         *
         * Getter for ProgramOptions populated by the arguement parser
         * @returns ProgramOptions structure.
        */
        Markov::API::CLI::ProgramOptions getProgramOptions(void) {
            return this->po;
        }

        /** @brief Initialize program options structure.
         *
         * @param i boolean, true if import operation is flagged
         * @param e boolean, true if export operation is flagged
         * @param bf boolean, true if there is something wrong with the command line parameters
         * @param s seperator character for the import function
         * @param iName import filename
         * @param exName export filename
         * @param oName output filename
         * @param dName corpus filename
         * @param n number of passwords to be generated
         *
        */
        void setProgramOptions(bool i, bool e, bool bf, char s, std::string iName, std::string exName,
      std::string oName, std::string dName, int n) {
            this->po.bImport = i;
            this->po.bExport = e;
            this->po.seperator = s;
            this->po.bFailure = bf;
            this->po.generateN = n;
            this->po.importname = iName;
            this->po.exportname = exName;
            this->po.outputfilename = oName;
            this->po.datasetname = dName;

            /*strcpy_s(this->po.importname,256,iName);
            strcpy_s(this->po.exportname,256,exName);
            strcpy_s(this->po.outputfilename,256,oName);
            strcpy_s(this->po.datasetname,256,dName);*/

        }

        /** @brief parse cli commands and return
        * @param argc - Program arguement count
        * @param argv - Program arguement values array
        * @return ProgramOptions structure.
        */
        static Markov::API::CLI::ProgramOptions* parse(int argc, char** argv);


        /** @brief Print help string.
        */
        static void help();

    private:
        Markov::API::CLI::ProgramOptions po;
    };

};
```

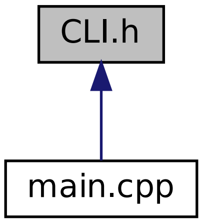
9.7 CLI.h File Reference

```
#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

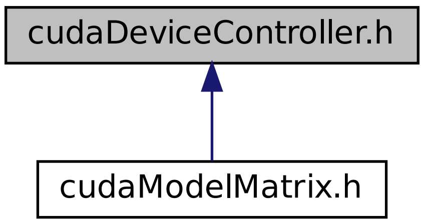
  namespace for MarkovPasswords API GUI wrapper

9.8 CLI.h

```
#pragma once
#include <QtWidgets/QMainWindow>
#include "ui_CLI.h"

namespace Markov::GUI{
    /** @brief QT CLI Class
    */
    class CLI :public QMainWindow {
        Q_OBJECT
    public:
        CLI(QWidget* parent = Q_NULLPTR);

    private:
        Ui::CLI ui;

    public slots:
        void start();
        void statistics();
        void about();
    };
};
```

9.9 cudaDeviceController.h File Reference

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



Classes

- class Markov::API::CUDA::CUDADeviceController

    Controller class for CUDA device.

Namespaces

- Markov

    Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

    Namespace for the MarkovPasswords API.
- Markov::API::CUDA

    Namespace for objects requiring CUDA libraries.

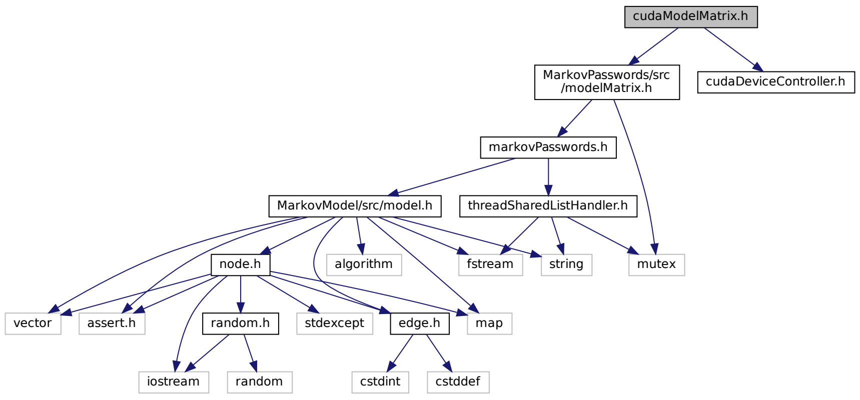
9.10 cudaDeviceController.h

```

/** @brief Namespace for objects requiring CUDA libraries.
*/
namespace Markov::API::CUDA{
    /** @brief Controller class for CUDA device
     *
     * This implementation only supports Nvidia devices.
    */
    class CUDADeviceController{
    public:
        /** @brief List CUDA devices in the system.
         *
         * This function will print details of every CUDA capable device in the system.
         *
         * @b Example @b output:
         * @code{.txt}
         * Device Number: 0
         * Device name: GeForce RTX 2070
         * Memory Clock Rate (KHz): 7001000
         * Memory Bus Width (bits): 256
         * Peak Memory Bandwidth (GB/s): 448.064
         * Max Linear Threads: 1024
         * @endcode
        */
        __host__ void ListCudaDevices();

    protected:
        /** @brief Check results of the last operation on GPU.
         *
         * Check the status returned from cudaMalloc/cudaMemcpy to find failures.
         *
         * If a failure occurs, its assumed beyond redemption, and exited.
         * @param _status Cuda error status to check
         * @param msg Message to print in case of a failure
         * @return 0 if successful, 1 if failure.
         * @b Example @b output:
         * @code{.txt}
         * char *da, a = "test";
         * cudastatus = cudaMalloc((char **)&da, 5*sizeof(char*));
         * CudaCheckNotifyErr(cudastatus, "Failed to allocate VRAM for *da.\n");
         * @endcode
        */
        __host__ int CudaCheckNotifyErr(cudaError_t _status, const char* msg);

    private:
    };
};
```

9.11 cudaModelMatrix.h File Reference

```
#include "MarkovPasswords/src/modelMatrix.h"
#include "cudaDeviceController.h"
```

Include dependency graph for cudaModelMatrix.h:



Classes

- class Markov::API::CUDA::CUDAModelMatrix

  Extension of Markov::API::ModelMatrix which is modified to run on GPU devices.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.
- Markov::API::CUDA

  Namespace for objects requiring CUDA libraries.

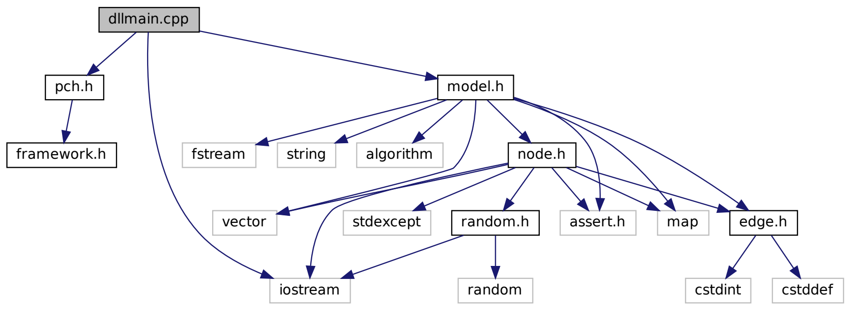
9.12 cudaModelMatrix.h

```
#include "MarkovPasswords/src/modelMatrix.h"
#include "cudaDeviceController.h"

/** @brief Namespace for objects requiring CUDA libraries.
*/
namespace Markov::API::CUDA{
    /** @brief Extension of Markov::API::ModelMatrix which is modified to run on GPU devices.
     *
     * This implementation only supports Nvidia devices.
    */
    class CUDAModelMatrix : public ModelMatrix, public CUDADeviceController{
    public:

        /** @brief Migrate the class members to the VRAM
         *
         * Cannot be used without calling Markov::API::ModelMatrix::ConstructMatrix at least once.
         * This function will manage the memory allocation and data transfer from CPU RAM to GPU VRAM.
         *
         * Newly allocated VRAM pointers are set in the class member variables.
         *
        */
        __host__ void MigrateMatrix();

        /** @brief Random walk on the Matrix-reduced Markov::Model
         *
         * TODO
         *
         *
         * @param n - Number of passwords to generate.
         * @param wordlistFileName - Filename to write to
         * @param minLen - Minimum password length to generate
         * @param maxLen - Maximum password length to generate
         * @param threads - number of OS threads to spawn
         * @param bFileIO - If false, filename will be ignored and will output to stdout.
         *
         *
         * @code{.cpp}
         * Markov::API::ModelMatrix mp;
         * mp.Import("models/finished.mdl");
         * mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
         * @endcode
         *
        */
        __host__ void FastRandomWalk(unsigned long int n, const char* wordlistFileName, int minLen,
 int maxLen, int threads, bool bFileIO);

    protected:
        /** @brief Retrieve the result buffer from CUDA kernel.
         *
         * Done on each partition.
         *
         *
        */
        __host__ void RetrieveCudaBuffer(/*TODO*/);


    private:
        char** device_edgeMatrix;
        long int **device_valueMatrix;
        char *device_matrixIndex;
        long int *device_totalEdgeWeights;
    };
};
```

9.13 dllmain.cpp File Reference

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

Include dependency graph for dllmain.cpp:



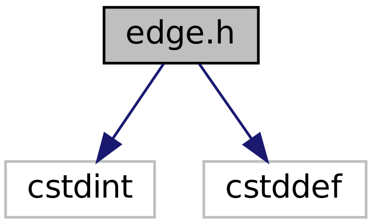
9.14 dllmain.cpp

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


#ifdef _WIN32
__declspec(dllexport) void dll_loadtest() {
    std::cout « "External function called.\n";
    //cudaTestEntry();
}

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}

#endif
```

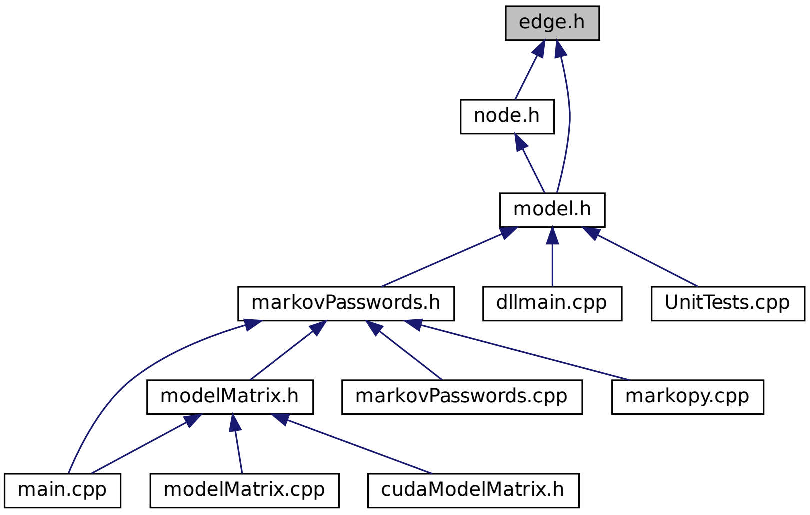
9.15 edge.h File Reference

`#include <cstdint>`
`#include <cstddef>`

Include dependency graph for edge.h:



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



Classes

- class Markov::Node< storageType >

  A node class that for the vertices of model. Connected with eachother using Edge.
- class Markov::Edge< NodeStorageType >

  Edge class used to link nodes in the model together.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

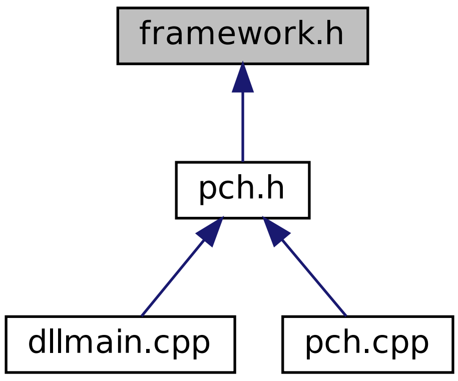
9.16 edge.h

```
#pragma once
#include <cstdint>
#include <cstddef>

namespace Markov {

    template <typename NodeStorageType>
    class Node;

    /** @brief 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.
    */
    template <typename NodeStorageType>
    class Edge {
    public:

        /** @brief Default constructor.
        */
        Edge<NodeStorageType>();

        /** @brief Constructor. Initialize edge with given RightNode and LeftNode
         * @param _left - Left node of this edge.
         * @param _right - Right node of this edge.
         *
         * @b Example @b Use: Construct edge
         * @code{.cpp}
         * 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);
         * @endcode
         *
        */
        Edge<NodeStorageType>(Node<NodeStorageType>* _left, Node<NodeStorageType>* _right);

        /** @brief Adjust the edge EdgeWeight with offset.
         * Adds the offset parameter to the edge EdgeWeight.
         * @param offset - NodeValue to be added to the EdgeWeight
         *
         * @b Example @b Use: Construct edge
         * @code{.cpp}
         * 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);
         *
         * @endcode
        */
        void AdjustEdge(long int offset);

        /** @brief Traverse this edge to RightNode.
         * @return Right node. If this is a terminator node, return NULL
         *
         *
         * @b Example @b Use: Traverse a node
         * @code{.cpp}
         * 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();
         * @endcode
         *
        */
        inline Node<NodeStorageType>* TraverseNode();

        /** @brief Set LeftNode of this edge.
        * @param node - Node to be linked with.
        */
        void SetLeftEdge (Node<NodeStorageType>*);
        /** @brief Set RightNode of this edge.
        * @param node - Node to be linked with.
        */
        void SetRightEdge(Node<NodeStorageType>*);

        /** @brief return edge's EdgeWeight.
        * @return edge's EdgeWeight.
        */
        inline uint64_t EdgeWeight();

        /** @brief return edge's LeftNode
        * @return edge's LeftNode.
        */
        Node<NodeStorageType>* LeftNode();

        /** @brief return edge's RightNode
        * @return edge's RightNode.
        */
        inline Node<NodeStorageType>* RightNode();

    private:
        Node<NodeStorageType>* _left; /** @brief source node*/
        Node<NodeStorageType>* _right;/** @brief target node*/
        long int _weight;    /** @brief Edge EdgeWeight*/
    };


};

//default constructor of edge
template <typename NodeStorageType>
Markov::Edge<NodeStorageType>::Edge() {
    this->_left = NULL;
    this->_right = NULL;
    this->_weight = 0;
}
//constructor of edge
template <typename NodeStorageType>
Markov::Edge<NodeStorageType>::Edge(Markov::Node<NodeStorageType>* _left,
 Markov::Node<NodeStorageType>* _right) {
    this->_left = _left;
    this->_right = _right;
    this->_weight = 0;
```

```
}
//to AdjustEdge the edges by the edge with its offset
template <typename NodeStorageType>
void Markov::Edge<NodeStorageType>::AdjustEdge(long int offset) {
    this->_weight += offset;
    this->LeftNode()->UpdateTotalVerticeWeight(offset);
}
//to TraverseNode the node
template <typename NodeStorageType>
inline Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::TraverseNode() {
    if (this->RightNode()->NodeValue() == 0xff) //terminator node
        return NULL;
    return _right;
}
//to set the LeftNode of the node
template <typename NodeStorageType>
void Markov::Edge<NodeStorageType>::SetLeftEdge(Markov::Node<NodeStorageType>* n) {
    this->_left = n;
}
//to set the RightNode of the node
template <typename NodeStorageType>
void Markov::Edge<NodeStorageType>::SetRightEdge(Markov::Node<NodeStorageType>* n) {
    this->_right = n;
}
//to get the EdgeWeight of the node
template <typename NodeStorageType>
inline uint64_t Markov::Edge<NodeStorageType>::EdgeWeight() {
    return this->_weight;
}
//to get the LeftNode of the node
template <typename NodeStorageType>
Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::LeftNode() {
    return this->_left;
}
//to get the RightNode of the node
template <typename NodeStorageType>
inline Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::RightNode() {
    return this->_right;
}
```

9.17 framework.h File Reference

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



Macros

- #define WIN32_LEAN_AND_MEAN

9.17.1 Macro Definition Documentation

9.17.1.1 WIN32_LEAN_AND_MEAN

```
#define WIN32_LEAN_AND_MEAN
```

Definition at line 3 of file framework.h.

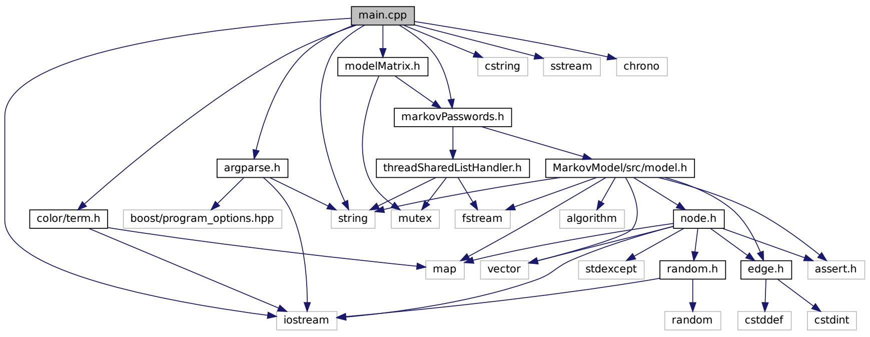
9.18 framework.h

```
00001 #pragma once
00002
00003 #define WIN32_LEAN_AND_MEAN             // Exclude rarely-used stuff from Windows headers
00004 // Windows Header Files
00005
00006 #ifdef _WIN32
00007 #include <windows.h>
00008 #endif
```

9.19 main.cpp File Reference

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

Include dependency graph for src/main.cpp:



Functions

- int main (int argc, char **argv)

  Launch CLI tool.

9.19.1 Function Documentation

9.19.1.1 main()

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

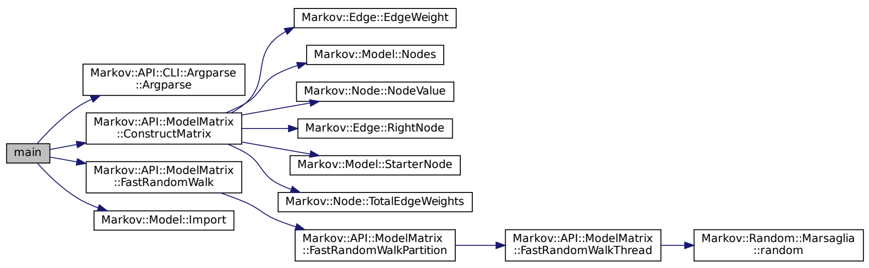
Launch CLI tool.

Definition at line 14 of file src/main.cpp.

```
00014                                       {
00015
00016     Markov::API::CLI::Terminal t;
00017     /*
00018     ProgramOptions* p  = Argparse::parse(argc, argv);
00019
00020     if (p==0 || p->bFailure) {
00021         std::cout « TERM_FAIL « "Arguments Failed to Parse" « std::endl;
00022         Argparse::help();
00023     }*/
00024     Markov::API::CLI::Argparse a(argc,argv);
00025
00026     Markov::API::ModelMatrix markovPass;
00027     std::cerr « "Importing model.\n";
00028     markovPass.Import("models/finished.mdl");
00029     std::cerr « "Import done. \n";
00030     markovPass.ConstructMatrix();
00031     std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
00032     //markovPass.FastRandomWalk(50000000,"/media/ignis/Stuff/wordlist.txt",6,12,25, true);
00033     markovPass.FastRandomWalk(500000000,"/media/ignis/Stuff/wordlist2.txt",6,12,25, true);
00034     std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
00035
00036     std::cerr « "Finished in:" « std::chrono::duration_cast<std::chrono::milliseconds> (end -
       begin).count() « " milliseconds" « std::endl;
00037     return 0;
00038 }
```

References Markov::API::CLI::Argparse::Argparse(), Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::FastRa
and Markov::Model< NodeStorageType >::Import().

Here is the call graph for this function:


main
Markov::API::CLI::Argparse ::Argparse
Markov::API::ModelMatrix ::ConstructMatrix
Markov::API::ModelMatrix ::FastRandomWalk
Markov::Model::Import
Markov::Edge::EdgeWeight
Markov::Model::Nodes
Markov::Node::NodeValue
Markov::Edge::RightNode
Markov::Model::StarterNode
Markov::Node::TotalEdgeWeights
Markov::API::ModelMatrix ::FastRandomWalkPartition
Markov::API::ModelMatrix ::FastRandomWalkThread
Markov::Random::Marsaglia ::random

9.20 src/main.cpp

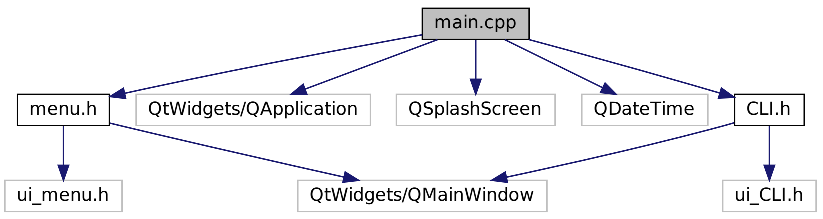
```
00001 #pragma once
00002 #include <iostream>
00003 #include "color/term.h"
00004 #include "argparse.h"
00005 #include <string>
00006 #include <cstring>
00007 #include <sstream>
00008 #include "markovPasswords.h"
00009 #include "modelMatrix.h"
00010 #include <chrono>
00011
00012 /** @brief Launch CLI tool.
00013 */
00014 int main(int argc, char** argv) {
00015
00016     Markov::API::CLI::Terminal t;
00017     /*
00018     ProgramOptions* p  = Argparse::parse(argc, argv);
00019
00020     if (p==0 || p->bFailure) {
00021         std::cout « TERM_FAIL « "Arguments Failed to Parse" « std::endl;
00022         Argparse::help();
00023     }*/
00024     Markov::API::CLI::Argparse a(argc,argv);
00025
00026     Markov::API::ModelMatrix markovPass;
00027     std::cerr « "Importing model.\n";
00028     markovPass.Import("models/finished.mdl");
00029     std::cerr « "Import done. \n";
```

```
    markovPass.ConstructMatrix();
    std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
    //markovPass.FastRandomWalk(50000000,"/media/ignis/Stuff/wordlist.txt",6,12,25, true);
    markovPass.FastRandomWalk(500000000,"/media/ignis/Stuff/wordlist2.txt",6,12,25, true);
    std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();

    std::cerr « "Finished in:" « std::chrono::duration_cast<std::chrono::milliseconds> (end -
begin).count() « " milliseconds" « std::endl;
    return 0;
}
```

9.21 main.cpp File Reference

```
#include "menu.h"
#include <QtWidgets/QApplication>
#include <QSplashScreen>
#include <QDateTime>
#include "CLI.h"
```

Include dependency graph for UI/src/main.cpp:



Functions

- int main (int argc, char ∗argv[ ])

  Launch UI.

9.21.1 Function Documentation

9.21.1.1 main()

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

Launch UI.

Definition at line 12 of file UI/src/main.cpp.

```
{



    QApplication a(argc, argv);

    QPixmap loadingPix("views/startup.jpg");
    QSplashScreen splash(loadingPix);
    splash.show();
    QDateTime time = QDateTime::currentDateTime();
    QDateTime currentTime = QDateTime::currentDateTime();   //Record current time
    while (time.secsTo(currentTime) <= 5)                   //5 is the number of seconds to delay
    {
        currentTime = QDateTime::currentDateTime();
        a.processEvents();
    };


    CLI w;
    w.show();
    splash.finish(&w);
    return a.exec();
}
```

References Markov::GUI::CLI::start().

Here is the call graph for this function:

main
Markov::GUI::CLI::start

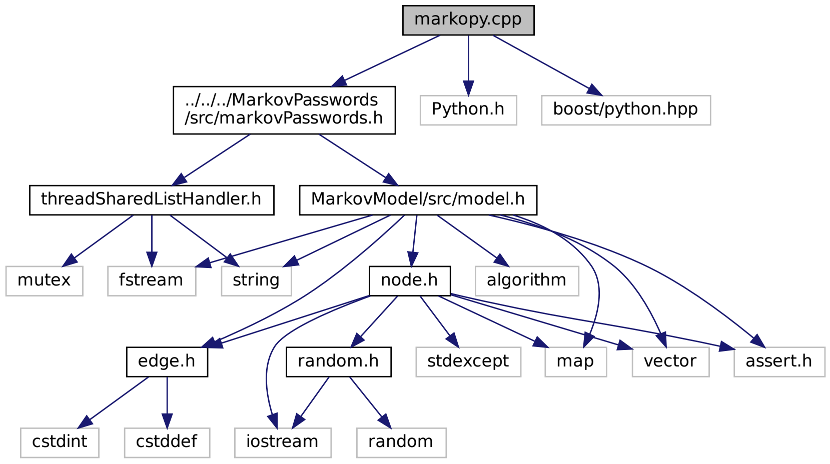
9.22 UI/src/main.cpp

```
//#include "MarkovPasswordsGUI.h"
#include "menu.h"
#include <QtWidgets/QApplication>
#include <QSplashScreen>
#include < QDateTime >
#include "CLI.h"

using namespace Markov::GUI;

/** @brief Launch UI.
*/
int main(int argc, char *argv[])
{



    QApplication a(argc, argv);

    QPixmap loadingPix("views/startup.jpg");
    QSplashScreen splash(loadingPix);
    splash.show();
    QDateTime time = QDateTime::currentDateTime();
    QDateTime currentTime = QDateTime::currentDateTime();   //Record current time
    while (time.secsTo(currentTime) <= 5)                   //5 is the number of seconds to delay
    {
        currentTime = QDateTime::currentDateTime();
        a.processEvents();
    };


    CLI w;
    w.show();
    splash.finish(&w);
    return a.exec();
}
```

9.23 markopy.cpp File Reference

```
#include "../../../MarkovPasswords/src/markovPasswords.h"
#include <Python.h>
#include <boost/python.hpp>
```

Include dependency graph for markopy.cpp:



Namespaces

- Markov

    Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::Markopy

Macros

- #define BOOST_PYTHON_STATIC_LIB

Functions

- Markov::Markopy::BOOST_PYTHON_MODULE (markopy)

9.23.1 Macro Definition Documentation

9.23.1.1 BOOST_PYTHON_STATIC_LIB

```
#define BOOST_PYTHON_STATIC_LIB
```

Definition at line 4 of file markopy.cpp.

9.24 markopy.cpp

```
00001 #pragma once
00002 #include "../../../MarkovPasswords/src/markovPasswords.h"
00003
00004 #define BOOST_PYTHON_STATIC_LIB
00005 #include <Python.h>
00006 #include <boost/python.hpp>
00007
00008 using namespace boost::python;
00009
00010 namespace Markov::Markopy{
00011     BOOST_PYTHON_MODULE(markopy)
00012     {
00013         bool (Markov::API::MarkovPasswords::*Import)(const char*) = &Markov::Model<char>::Import;
00014         bool (Markov::API::MarkovPasswords::*Export)(const char*) = &Markov::Model<char>::Export;
00015         class_<Markov::API::MarkovPasswords>("MarkovPasswords", init<>())
00016             .def(init<>())
```

```
            .def("Train", &Markov::API::MarkovPasswords::Train,
            "Train the model\n"
            "\n"
            ":param datasetFileName: Ifstream* to the dataset. If null, use class member\n"
            ":param delimiter: a character, same as the delimiter in dataset content\n"
            ":param threads: number of OS threads to spawn\n")
            .def("Generate", &Markov::API::MarkovPasswords::Generate,
            "Generate passwords from a trained model.\n"
            ":param n: Ifstream* to the dataset. If null, use class member\n"
            ":param wordlistFileName: a character, same as the delimiter in dataset content\n"
            ":param minLen: number of OS threads to spawn\n"
            ":param maxLen: Ifstream* to the dataset. If null, use class member\n"
            ":param threads: a character, same as the delimiter in dataset content\n"
            ":param threads: number of OS threads to spawn\n")
            .def("Import", Import, "Import a model file.")
            .def("Export", Export, "Export a model to file.")
        ;
    };
};
```

9.25 markopy_cli.py File Reference

Namespaces

- markopy_cli

Functions

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

Variables

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

9.26 markopy_cli.py

```
#!/usr/bin/python3
"""
  @namespace Markov::Markopy::Python
"""

import markopy
import argparse
import allogate as logging
import re
import os

parser = argparse.ArgumentParser(description="Python wrapper for MarkovPasswords.",
                                 epilog=f"""Sample runs:
{__file__} train untrained.mdl -d dataset.dat -s "\\t" -o trained.mdl
```

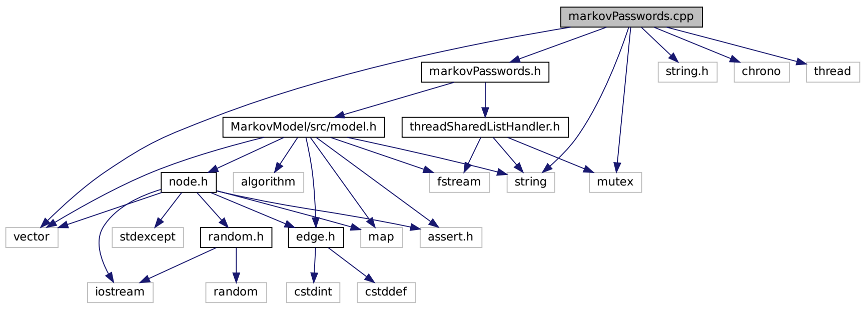
```
    Import untrained.mdl, train it with dataset.dat which has tab delimited data, output resulting model to trained.mdl\n

{__file__} generate trained.mdl -n 500 -w output.txt
    Import trained.mdl, and generate 500 lines to output.txt

{__file__} combine untrained.mdl -d dataset.dat -s "\\t" -n 500 -w output.txt
    Train and immediately generate 500 lines to output.txt. Do not export trained model.

{__file__} combine untrained.mdl -d dataset.dat -s "\\t" -n 500 -w output.txt -o trained.mdl
    Train and immediately generate 500 lines to output.txt. Export trained model.
""", formatter_class=argparse.RawTextHelpFormatter)

parser.add_argument("mode", help="Operation mode, supported modes: \"generate\", \"train\" and \"combine\".")
parser.add_argument("input", help="Input model file. This model will be imported before starting operation.\n"
                                  + "For more information on the file structure for input, check out the wiki page.")
parser.add_argument("-o", "--output",
                    help="Output model filename. This model will be exported when done. Will be ignored for generation mode.")
parser.add_argument("-d", "--dataset",
                    help="Dataset filename to read input from for training. Will be ignored for generation mode.\n"
                    + "Dataset is occurrence of a string and the string value seperated by a seperator. For more info "
                    + "on the dataset file structure, check out the github wiki page.")
parser.add_argument("-s", "--seperator",
                    help="Seperator character to use with training data.(character between occurrence and value)\n"
                    + "For more information on dataset/corpus file structure, check out the github wiki.")
parser.add_argument("-w", "--wordlist",
                    help="Wordlist filename path to export generation results to. Will be ignored for training mode")
parser.add_argument("--min", default=6, help="Minimum length that is allowed during generation.\n"
                    + "Any string shorter than this paremeter will retry to continue instead of proceeding to "
                    + "finishing node")
parser.add_argument("--max", default=12, help="Maximum length that is allowed during generation.\n"
                    +"Any string that does reaches this length are cut off irregardless to their position on the model.")
parser.add_argument("-n", "--count", help="Number of lines to generate. Ignored in training mode.")
parser.add_argument("-t", "--threads", default=10, help="Number of threads to use with training/generation.\n"
                    +"This many OS threads will be created for training/generation functions")
parser.add_argument("-v", "--verbosity", action="count", help="Output verbosity.\n"
                    + "Set verbosity to 1: -v\n"
                    + "Set verbosity to 3: -vvv\n"
                    + "Print pretty much everything, including caller functions: -vvvvvvvvvvvvvvv")
parser.add_argument("-b", "--bulk", action="store_true",
                    help="Bulk generate or bulk train every corpus/model in the folder.\n"
                    + "If working on this mode, output/input/dataset parameters should be a folder.\n"
                    + "Selected operation (generate/train) will be applied to each file in the folder, and "
                    + "output to the output directory.")
args = parser.parse_args()


def cli_init(input_model):
    logging.VERBOSITY = 0
    if args.verbosity:
        logging.VERBOSITY = args.verbosity
        logging.pprint(f"Verbosity set to {args.verbosity}.", 2)

    logging.pprint("Initializing model.", 1)
    model = markopy.MarkovPasswords()
    logging.pprint("Model initialized.", 2)

    logging.pprint("Importing model file.", 1)

    if (not os.path.isfile(input_model)):
        logging.pprint(f"Model file at {input_model} not found. Check the file path, or working directory")
        exit(1)

    model.Import(input_model)
    logging.pprint("Model imported successfully.", 2)
    return model


def cli_train(model, dataset, seperator, output, output_forced=False, bulk=False):
    if not (dataset and seperator and (output or not output_forced)):
        logging.pprint(
            f"Training mode requires -d/--dataset{', -o/--output' if output_forced else ''} and -s/--seperator parameters. Exiting.")
        exit(2)

    if (not bulk and not os.path.isfile(dataset)):
        logging.pprint(f"{dataset} doesn’t exists. Check the file path, or working directory")
        exit(3)

    if (output and os.path.isfile(output)):
        logging.pprint(f"{output} exists and will be overwritten.", 1)

    if (seperator == ’\\t’):
        logging.pprint("Escaping seperator.", 3)
        seperator = ’\t’

    if (len(seperator) != 1):
        logging.pprint(f’Delimiter must be a single character, and "{seperator}" is not accepted.’)
        exit(4)

    logging.pprint(f’Starting training.’, 3)
    model.Train(dataset, seperator, int(args.threads))
    logging.pprint(f’Training completed.’, 2)

    if (output):
        logging.pprint(f’Exporting model to {output}’, 2)
        model.Export(output)
    else:
        logging.pprint(f’Model will not be exported.’, 1)


def cli_generate(model, wordlist, bulk=False):
    if not (wordlist or args.count):
        logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters. Exiting.")
        exit(2)

    if (bulk and os.path.isfile(wordlist)):
        logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
    model.Generate(int(args.count), wordlist, int(args.min), int(args.max), int(args.threads))


if (args.bulk):
    logging.pprint(f"Bulk mode operation chosen.", 4)

    if (args.mode.lower() == "train"):
        if (os.path.isdir(args.output) and not os.path.isfile(args.output)) and (
                os.path.isdir(args.dataset) and not os.path.isfile(args.dataset)):
            corpus_list = os.listdir(args.dataset)
            for corpus in corpus_list:
                model = cli_init(args.input)
                logging.pprint(f"Training {args.input} with {corpus}", 2)
                output_file_name = corpus
                model_extension = ""
                if "." in args.input:
                    model_extension = args.input.split(".")[-1]
                cli_train(model, f"{args.dataset}/{corpus}", args.seperator,
                          f"{args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
        else:
            logging.pprint("In bulk training, output and dataset should be a directory.")
            exit(1)

    elif (args.mode.lower() == "generate"):
        if (os.path.isdir(args.wordlist) and not os.path.isfile(args.wordlist)) and (
                os.path.isdir(args.input) and not os.path.isfile(args.input)):
            model_list = os.listdir(args.input)
            print(model_list)
            for input in model_list:
                logging.pprint(f"Generating from {args.input}/{input} to {args.wordlist}/{input}.txt",
 2)

                model = cli_init(f"{args.input}/{input}")
                model_base = input
                if "." in args.input:
                    model_base = input.split(".")[1]
                cli_generate(model, f"{args.wordlist}/{model_base}.txt", bulk=True)
        else:
            logging.pprint("In bulk generation, input and wordlist should be directory.")

else:
    model = cli_init(args.input)
    if (args.mode.lower() == "generate"):
        cli_generate(model, args.wordlist)


    elif (args.mode.lower() == "train"):
        cli_train(model, args.dataset, args.seperator, args.output, output_forced=True)


    elif (args.mode.lower() == "combine"):
        cli_train(model, args.dataset, args.seperator, args.output)
```

```
            cli_generate(model, args.wordlist)


    else:
        logging.pprint("Invalid mode arguement given.")
        logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
        exit(5)
```

9.27 markovPasswords.cpp File Reference

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

Include dependency graph for markovPasswords.cpp:



9.28 markovPasswords.cpp

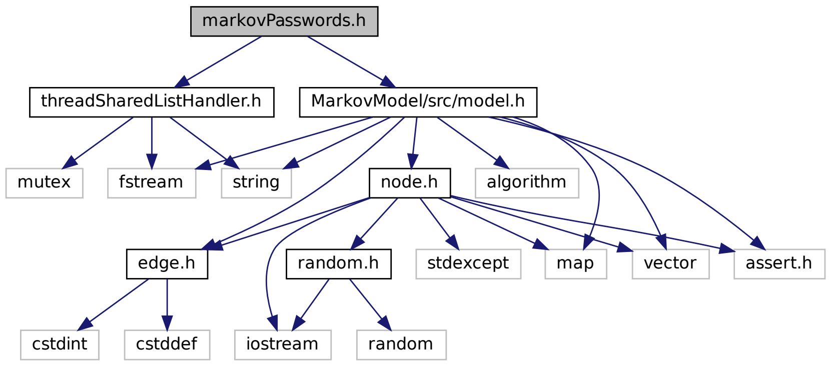
```
#pragma once
#include "markovPasswords.h"
#include <string.h>
#include <chrono>
#include <thread>
#include <vector>
#include <mutex>
#include <string>

Markov::API::MarkovPasswords::MarkovPasswords() : Markov::Model<char>(){


}

Markov::API::MarkovPasswords::MarkovPasswords(const char* filename) {

    std::ifstream* importFile;

    this->Import(filename);

    //std::ifstream* newFile(filename);

    //importFile = newFile;

}

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

    std::ifstream* datasetFile;

    std::ifstream newFile(filename);

    datasetFile = &newFile;

    this->Import(datasetFile);
    return datasetFile;
```

```
}


void Markov::API::MarkovPasswords::Train(const char* datasetFileName, char delimiter, int threads)   {
    Markov::API::Concurrency::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(&Markov::API::MarkovPasswords::TrainThread, this, &listhandler, 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 Markov::API::MarkovPasswords::TrainThread(Markov::API::Concurrency::ThreadSharedListHandler *listhandler, 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];
#ifdef _WIN32
        sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
#else
        sscanf(line.c_str(), format_str, &oc, linebuf);
#endif
        this->AdjustEdge((const char*)linebuf, oc);
        delete linebuf;
    }
}


std::ofstream* Markov::API::MarkovPasswords::Save(const char* filename) {
    std::ofstream* exportFile;

    std::ofstream newFile(filename);

    exportFile = &newFile;

    this->Export(exportFile);
    return exportFile;
}


void Markov::API::MarkovPasswords::Generate(unsigned long int n, const char* wordlistFileName, int minLen, int maxLen, int threads)  {
    char* res;
    char print[100];
    std::ofstream wordlist;
    wordlist.open(wordlistFileName);
    std::mutex mlock;
    int iterationsPerThread = n/threads;
    int iterationsCarryOver = n%threads;
    std::vector<std::thread*> threadsV;
    for(int i=0;i<threads;i++){
        threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this, &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
    }

    for(int i=0;i<threads;i++){
        threadsV[i]->join();
        delete threadsV[i];
    }

    this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);

}

void Markov::API::MarkovPasswords::GenerateThread(std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)  {
    char* res = new char[maxLen+5];
    if(n==0) return;

    Markov::Random::Marsaglia MarsagliaRandomEngine;
    for (int i = 0; i < n; i++) {
        this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
        outputLock->lock();
        *wordlist « res « "\n";
        outputLock->unlock();
    }
}
```

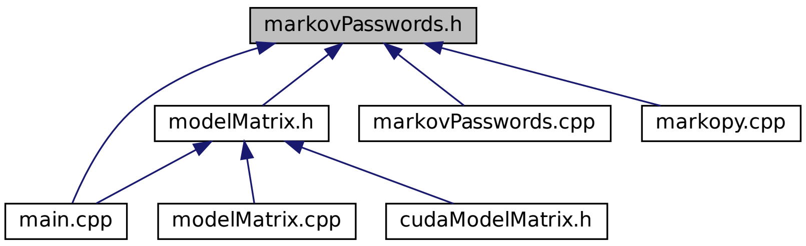
9.29 markovPasswords.h File Reference

```
#include "threadSharedListHandler.h"
#include "MarkovModel/src/model.h"
```

Include dependency graph for markovPasswords.h:



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



Classes

- class Markov::API::MarkovPasswords

  Markov::Model with char represented nodes.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.

9.30 markovPasswords.h

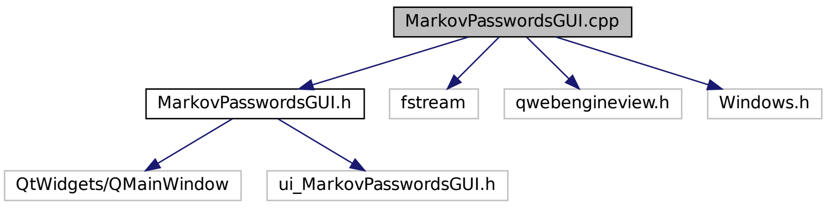
```
#pragma once
#include "threadSharedListHandler.h"
#include "MarkovModel/src/model.h"


/** @brief Namespace for the MarkovPasswords API
*/
namespace Markov::API{

    /** @brief Markov::Model with char represented nodes.
     *
     * Includes wrappers for Markov::Model and additional helper functions to handle file I/O
     *
     * This class is an extension of Markov::Model<char>, with higher level abstractions such as train
 and generate.
     *
    */
    class MarkovPasswords : public Markov::Model<char>{
    public:

        /** @brief Initialize the markov model from MarkovModel::Markov::Model.
         *
         * Parent constructor. Has no extra functionality.
        */
        MarkovPasswords();

        /** @brief 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.
         *
         * @param filename - Filename to import
         *
         *
         * @b Example @b Use: Construction via filename
         * @code{.cpp}
         * MarkovPasswords mp("test.mdl");
         * @endcode
        */
        MarkovPasswords(const char* filename);

        /** @brief Open dataset file and return the ifstream pointer
         * @param filename - Filename to open
         * @return ifstream* to the the dataset file
        */
        std::ifstream* OpenDatasetFile(const char* filename);


        /** @brief Train the model with the dataset file.
         * @param datasetFileName - Ifstream* to the dataset. If null, use class member
         * @param delimiter - a character, same as the delimiter in dataset content
         * @param threads - number of OS threads to spawn
         *
         * @code{.cpp}
         * Markov::API::MarkovPasswords mp;
         * mp.Import("models/2gram.mdl");
         * mp.Train("password.corpus");
         * @endcode
        */
        void Train(const char* datasetFileName, char delimiter, int threads);



        /** @brief Export model to file.
        * @param filename - Export filename.
        * @return std::ofstream* of the exported file.
        */
        std::ofstream* Save(const char* filename);

        /** @brief Call Markov::Model::RandomWalk n times, and collect output.
         *
         * Generate from model and write results to a file.
         * a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5
 on average.
         *
         * @deprecated See Markov::API::MatrixModel::FastRandomWalk for more information.
         * @param n - Number of passwords to generate.
         * @param wordlistFileName - Filename to write to
         * @param minLen - Minimum password length to generate
         * @param maxLen - Maximum password length to generate
         * @param threads - number of OS threads to spawn
        */
        void Generate(unsigned long int n, const char* wordlistFileName, int minLen=6, int maxLen=12,
 int threads=20);
```

```


    private:

        /** @brief A single thread invoked by the Train function.
         * @param listhandler - Listhandler class to read corpus from
         * @param delimiter - a character, same as the delimiter in dataset content
         *
        */
        void TrainThread(Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char
delimiter);

        /** @brief A single thread invoked by the Generate function.
         *
         * @b DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information.
This has been replaced with
         * a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5
on average.
         *
         * @param outputLock - shared mutex lock to lock during output operation. Prevents race
condition on write.
         * @param n number of lines to be generated by this thread
         * @param wordlist wordlistfile
         * @param minLen - Minimum password length to generate
         * @param maxLen - Maximum password length to generate
         *
        */
        void GenerateThread(std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int
minLen, int maxLen);
        std::ifstream* datasetFile;
        std::ofstream* modelSavefile;
        std::ofstream* outputFile;
    };



};
```

9.31 MarkovPasswordsGUI.cpp File Reference

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

Include dependency graph for MarkovPasswordsGUI.cpp:



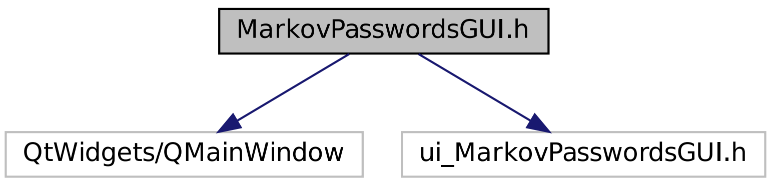
9.32 MarkovPasswordsGUI.cpp

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

using namespace Markov::GUI;

MarkovPasswordsGUI::MarkovPasswordsGUI(QWidget *parent)
    : QMainWindow(parent)
{
    ui.setupUi(this);


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


/*
Methods for buttons
*/

void MarkovPasswordsGUI::benchmarkSelected() {

    QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");

    //get working directory
    char path[255];
    GetCurrentDirectoryA(255, path);

    //get absolute path to the layout html
    std::string layout = "file:///" + std::string(path) + "\\views\\example.html";
    std::replace(layout.begin(), layout.end(), '\\', '/');
    webkit->setUrl(QUrl(layout.c_str()));
}


void MarkovPasswordsGUI::modelvisSelected() {

    QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");

    //get working directory
    char path[255];
    GetCurrentDirectoryA(255, path);

    //get absolute path to the layout html
    std::string layout = "file:///" + std::string(path) + "\\views\\model.htm";
    std::replace(layout.begin(), layout.end(), '\\', '/');
    webkit->setUrl(QUrl(layout.c_str()));
}

void MarkovPasswordsGUI::comparisonSelected() {

    QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");

    //get working directory
    char path[255];
    GetCurrentDirectoryA(255, path);

    //get absolute path to the layout html
    std::string layout = "file:///" + std::string(path) + "\\views\\comparison.htm";
    std::replace(layout.begin(), layout.end(), '\\', '/');
    webkit->setUrl(QUrl(layout.c_str()));
}





void MarkovPasswordsGUI::renderHTMLFile(std::string* filename) {
    //extract and parametrize the code from constructor

}



void MarkovPasswordsGUI::loadDataset(std::string* filename) {
    //extract and parametrize the code from constructor

}
```

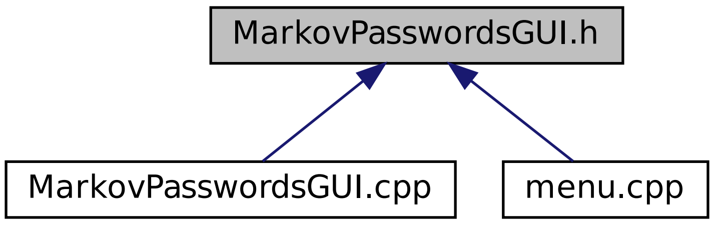
9.33 MarkovPasswordsGUI.h File Reference

```
#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

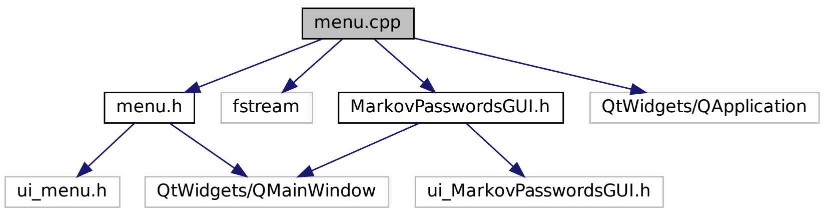
9.34 MarkovPasswordsGUI.h

```
#pragma once

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


namespace Markov::GUI{
    /** @brief Reporting UI.
    *
    * UI for reporting and debugging tools for MarkovPassword
    */
    class MarkovPasswordsGUI : public QMainWindow {
        Q_OBJECT

    public:
        /** @brief Default QT consturctor.
        * @param parent - Parent widget.
        */
        MarkovPasswordsGUI(QWidget *parent = Q_NULLPTR);

        /** @brief Render a HTML file.
        * @param filename - Filename of the html file. (relative path to the views folder).
        */
        void renderHTMLFile(std::string* filename);

        /** @brief Load a dataset to current view..
        * @param filename - Filename of the dataset file. (relative path to the views folder).
        */
        void loadDataset(std::string* filename);

    private:
        Ui::MarkovPasswordsGUIClass ui;


        //Slots for buttons in GUI.
    public slots:

        void MarkovPasswordsGUI::benchmarkSelected();
        void MarkovPasswordsGUI::modelvisSelected();
        void MarkovPasswordsGUI::visualDebugSelected();
        void MarkovPasswordsGUI::comparisonSelected();
    };
};
```

9.35 menu.cpp File Reference

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

Include dependency graph for menu.cpp:



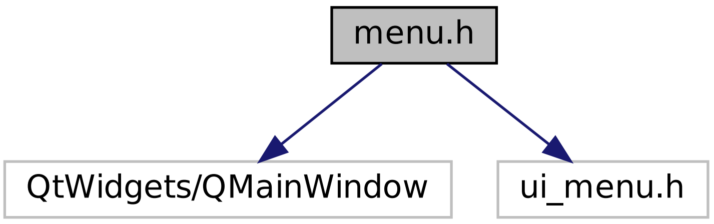
9.36 menu.cpp

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

using namespace Markov::GUI;

menu::menu(QWidget* parent)
    : QMainWindow(parent)
{
    ui.setupUi(this);


    //QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {about(); });
    QObject::connect(ui.visu, &QPushButton::clicked, this, [this] {visualization(); });
}
void menu::about() {


}
void menu::visualization() {
    MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
    w->show();
    this->close();
00025 }
```

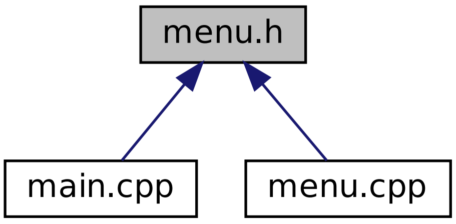
9.37 menu.h File Reference

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

Include dependency graph for menu.h:



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



Classes

- class Markov::GUI::menu

  QT Menu class.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

- Markov::GUI

  namespace for MarkovPasswords API GUI wrapper

9.38 menu.h

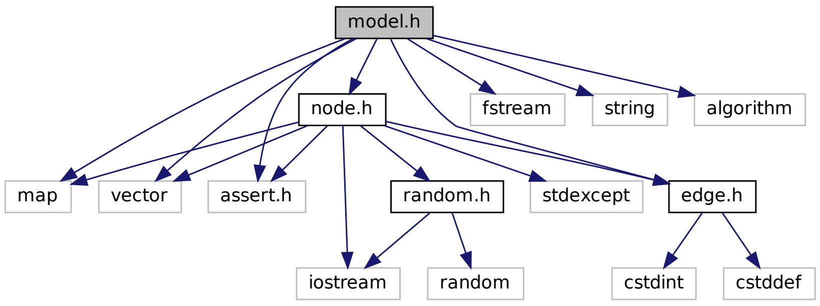
```
00001 #pragma once
00002 #include <QtWidgets/QMainWindow>
```

```
#include "ui_menu.h"


namespace Markov::GUI{
    /** @brief QT Menu class
    */
    class menu:public QMainWindow {
    Q_OBJECT
    public:
        menu(QWidget* parent = Q_NULLPTR);

    private:
        Ui::main ui;

    public slots:
        void about();
        void visualization();
    };
};
```

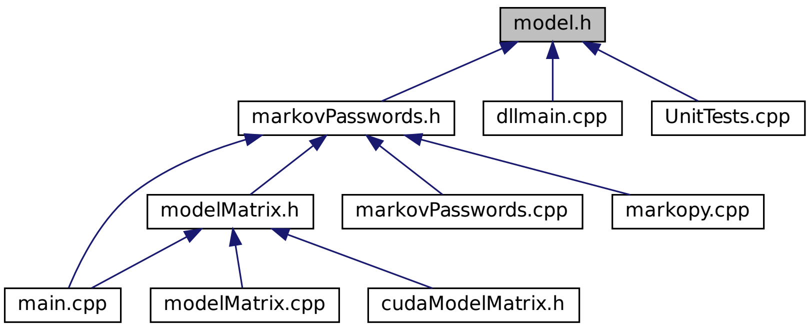
9.39 model.h File Reference

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

Include dependency graph for model.h:



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



Classes

- class Markov::Node< storageType >

  A node class that for the vertices of model. Connected with eachother using Edge.
- class Markov::Edge< NodeStorageType >

  Edge class used to link nodes in the model together.
- class Markov::Model< NodeStorageType >

  class for the final Markov Model, constructed from nodes and edges.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

9.40 model.h

```
/** @dir Model.h
 *
 */


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

/**
    @brief Namespace for the markov-model related classes.
    Contains Model, Node and Edge classes
*/
namespace Markov {

    template <typename NodeStorageType>
    class Node;

    template <typename NodeStorageType>
    class Edge;

    template <typename NodeStorageType>

    /** @brief 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.
     *
    */
    class Model {
    public:

        /** @brief 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.
        */
        Model<NodeStorageType>();

        /** @brief 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.
         *
         * @b Example @b Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use
Marsaglia
         * @code{.cpp}
         * 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";
         *  }
         * @endcode
         *
         * @param randomEngine Random Engine to use for the random walks. For examples, see
Markov::Random::Mersenne and Markov::Random::Marsaglia
         * @param minSetting Minimum number of characters to generate
         * @param maxSetting Maximum number of character to generate
         * @param buffer buffer to write the result to
         * @return Null terminated string that was generated.
        */
        NodeStorageType* RandomWalk(Markov::Random::RandomEngine* randomEngine, int minSetting, int
maxSetting, NodeStorageType* buffer);

        /** @brief 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.
         *
         * @b Example @b Use: Create an empty model and train it with string: "testdata"
         * @code{.cpp}
         * Markov::Model<char> model;
         * char test[] = "testdata";
         * model.AdjustEdge(test, 15);
         * @endcode
         *
         *
         * @param string - String that is passed from the training, and will be used to AdjustEdge the
model with
         * @param occurrence - Occurrence of this string.
         *
         *
        */
        void AdjustEdge(const NodeStorageType* payload, long int occurrence);

        /** @brief 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.
         * @return True if successful, False for incomplete models or corrupt file formats
         *
         * @b Example @b Use: Import a file from ifstream
         * @code{.cpp}
         * Markov::Model<char> model;
         * std::ifstream file("test.mdl");
         * model.Import(&file);
         * @endcode
        */
        bool Import(std::ifstream*);

        /** @brief Open a file to import with filename, and call bool Model::Import with std::ifstream
         * @return True if successful, False for incomplete models or corrupt file formats
         *
         * @b Example @b Use: Import a file with filename
         * @code{.cpp}
         * Markov::Model<char> model;
         * model.Import("test.mdl");
         * @endcode
        */
        bool Import(const char* filename);

        /** @brief 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.
         * @return True if successful, False for incomplete models.
         *
         * @b Example @b Use: Export file to ofstream
         * @code{.cpp}
         * Markov::Model<char> model;
         * std::ofstream file("test.mdl");
         * model.Export(&file);
         * @endcode
        */
        bool Export(std::ofstream*);

        /** @brief Open a file to export with filename, and call bool Model::Export with std::ofstream
         * @return True if successful, False for incomplete models or corrupt file formats
         *
         * @b Example @b Use: Export file to filename
         * @code{.cpp}
         * Markov::Model<char> model;
         * model.Export("test.mdl");
         * @endcode
        */
        bool Export(const char* filename);

        /** @brief Return starter Node
         * @return starter node with 00 NodeValue
        */
        Node<NodeStorageType>* StarterNode(){ return starterNode;}

        /** @brief Return a vector of all the edges in the model
         * @return vector of edges
        */
        std::vector<Edge<NodeStorageType>*>* Edges(){ return &edges;}

        /** @brief Return starter Node
         * @return starter node with 00 NodeValue
        */
        std::map<NodeStorageType, Node<NodeStorageType>*>* Nodes(){ return &nodes;}

    private:
        /** @brief Map LeftNode is the Nodes NodeValue
         * Map RightNode is the node pointer
        */
        std::map<NodeStorageType, Node<NodeStorageType>*> nodes;

        /** @brief Starter Node of this model.
         *
        */
        Node<NodeStorageType>* starterNode;


        /** @brief A list of all edges in this model.
         *
        */
        std::vector<Edge<NodeStorageType>*> edges;
    };

};

template <typename NodeStorageType>
Markov::Model<NodeStorageType>::Model() {
    this->starterNode = new Markov::Node<NodeStorageType>(0);
    this->nodes.insert({ 0, this->starterNode });
}

template <typename NodeStorageType>
bool Markov::Model<NodeStorageType>::Import(std::ifstream* f) {
    std::string cell;

    char src;
    char target;
    long int oc;

    while (std::getline(*f, cell)) {
        //std::cout « "cell: " « cell « std::endl;
        src = cell[0];
        target = cell[cell.length() - 1];
        char* j;
        oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
        //std::cout « oc « "\n";
        Markov::Node<NodeStorageType>* srcN;
        Markov::Node<NodeStorageType>* targetN;
        Markov::Edge<NodeStorageType>* e;
        if (this->nodes.find(src) == this->nodes.end()) {
            srcN = new Markov::Node<NodeStorageType>(src);
            this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
            //std::cout « "Creating new node at start.\n";
        }
        else {
            srcN = this->nodes.find(src)->second;
        }

        if (this->nodes.find(target) == this->nodes.end()) {
            targetN = new Markov::Node<NodeStorageType>(target);
            this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
            //std::cout « "Creating new node at end.\n";
        }
        else {
            targetN = this->nodes.find(target)->second;
        }
        e = srcN->Link(targetN);
        e->AdjustEdge(oc);
        this->edges.push_back(e);

        //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
       int(targetN->NodeValue()) « "\n";


    }

    for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
        //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
        std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
       Edge<NodeStorageType> *rhs)->bool{
            return lhs->EdgeWeight() > rhs->EdgeWeight();
        });
        //for(int i=0;i<x.second->edgesV.size();i++)
        //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
        //std::cout « "\n";
    }
    //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
    //std::cout « "Total number of edges: " « this->edges.size() « std::endl;

    return true;
}

template <typename NodeStorageType>
bool Markov::Model<NodeStorageType>::Import(const char* filename) {
    std::ifstream importfile;
    importfile.open(filename);
    return this->Import(&importfile);

}

template <typename NodeStorageType>
bool Markov::Model<NodeStorageType>::Export(std::ofstream* f) {
    Markov::Edge<NodeStorageType>* e;
    for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
        e = this->edges[i];
        //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
       e->RightNode()->NodeValue() « "\n";
        *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
       "\n";
    }

    return true;
}

template <typename NodeStorageType>
bool Markov::Model<NodeStorageType>::Export(const char* filename) {
    std::ofstream exportfile;
    exportfile.open(filename);
    return this->Export(&exportfile);
}

template <typename NodeStorageType>
NodeStorageType* Markov::Model<NodeStorageType>::RandomWalk(Markov::Random::RandomEngine*
       randomEngine, int minSetting, int maxSetting, NodeStorageType* buffer) {
    Markov::Node<NodeStorageType>* n = this->starterNode;
    int len = 0;
    Markov::Node<NodeStorageType>* 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();
    }

    //null terminate the string
    buffer[len] = 0x00;

    //do something with the generated string
    return buffer; //for now
}

template <typename NodeStorageType>
void Markov::Model<NodeStorageType>::AdjustEdge(const NodeStorageType* payload, long int occurrence) {
    NodeStorageType p = payload[0];
    Markov::Node<NodeStorageType>* curnode = this->starterNode;
    Markov::Edge<NodeStorageType>* e;
    int i = 0;

    if (p == 0) return;
    while (p != 0) {
        e = curnode->FindEdge(p);
        if (e == NULL) return;
        e->AdjustEdge(occurrence);
        curnode = e->RightNode();
        p = payload[++i];
    }

    e = curnode->FindEdge('\xff');
    e->AdjustEdge(occurrence);
    return;
}
```

9.41 model_2gram.py File Reference

Namespaces

- model_2gram

Variables

- model_2gram.alphabet = string.printable

  password alphabet

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

  output file handle

9.42 model_2gram.py

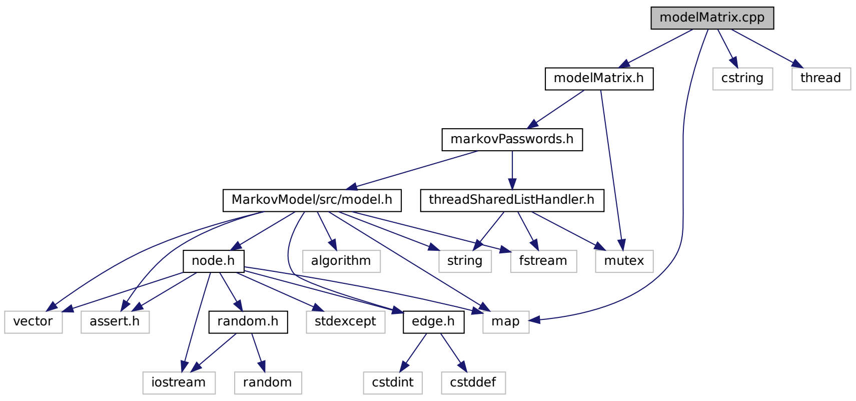
```
#!/usr/bin/python3
"""
  python script for generating a 2gram model
"""

import string
import re


alphabet = string.printable
alphabet = re.sub('\s', ”, alphabet)
print(f"alphabet={alphabet}")
#exit()


f = open('../../models/2gram.mdl', "wb")
#tie start nodes
for sym in alphabet:
    f.write(b"\x00,1," + bytes(sym, encoding='ascii') + b"\n")
```

```
#tie terminator nodes
for sym in alphabet:
    f.write(bytes(sym, encoding='ascii')+ b",1,\xff\n")

#tie internals
for src in alphabet:
    for target in alphabet:
        f.write(bytes(src, encoding='ascii') + b",1," + bytes(target, encoding='ascii') + b"\n")
```

9.43 modelMatrix.cpp File Reference

```
#include "modelMatrix.h"
#include <map>
#include <cstring>
#include <thread>
```

Include dependency graph for modelMatrix.cpp:



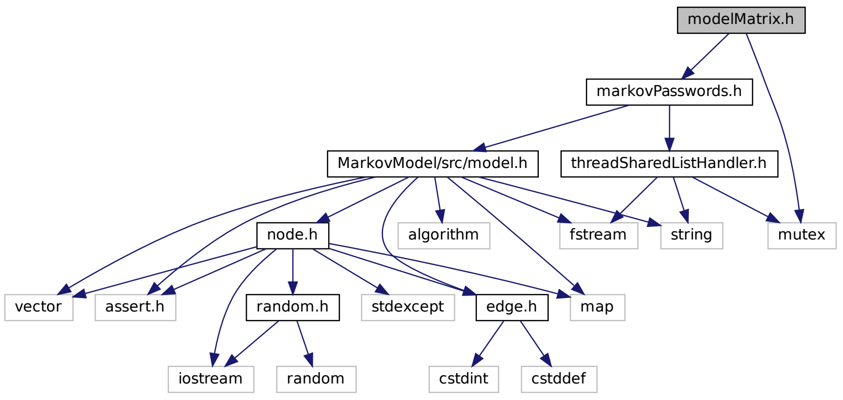
9.44 modelMatrix.cpp

```
#include "modelMatrix.h"
#include <map>
#include <cstring>
#include <thread>

Markov::API::ModelMatrix::ModelMatrix(){

}


void Markov::API::ModelMatrix::ConstructMatrix(){
    this->matrixSize = this->StarterNode()->edgesV.size() + 2;

    this->matrixIndex = new char[this->matrixSize];
    this->totalEdgeWeights = new long int[this->matrixSize];

    this->edgeMatrix = new char*[this->matrixSize];
    for(int i=0;i<this->matrixSize;i++){
        this->edgeMatrix[i] = new char[this->matrixSize];
    }
    this->valueMatrix = new long int*[this->matrixSize];
    for(int i=0;i<this->matrixSize;i++){
        this->valueMatrix[i] = new long int[this->matrixSize];
    }
    std::map< char, Node< char > * > *nodes;
    nodes = this->Nodes();
    int i=0;
    for (auto const& [repr, node] : *nodes){
        if(repr!=0) this->matrixIndex[i] = repr;
        else this->matrixIndex[i] = 199;
        this->totalEdgeWeights[i] = node->TotalEdgeWeights();
        for(int j=0;j<this->matrixSize;j++){
            char val = node->NodeValue();
            if(val < 0){
                for(int k=0;k<this->matrixSize;k++){
                    this->valueMatrix[i][k] = 0;
                    this->edgeMatrix[i][k] = 255;
                }
                break;
            }
            else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
                this->valueMatrix[i][j] = 0;
                this->edgeMatrix[i][j] = 255;
            }else if(j==(this->matrixSize-1)) {
                this->valueMatrix[i][j] = 0;
                this->edgeMatrix[i][j] = 255;
            }else{
                this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
                this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
            }

        }
        i++;
    }

    //this->DumpJSON();
}


void Markov::API::ModelMatrix::DumpJSON(){

    std::cout « "{\n   \"index\": \"";
    for(int i=0;i<this->matrixSize;i++){
        if(this->matrixIndex[i]=='"') std::cout « "\\\"";
        else if(this->matrixIndex[i]=='\\') std::cout « "\\\\";
        else if(this->matrixIndex[i]==0) std::cout « "\\\\x00";
        else if(i==0) std::cout « "\\\\xff";
        else if(this->matrixIndex[i]=='\n') std::cout « "\\n";
        else std::cout « this->matrixIndex[i];
    }
    std::cout «
    "\",\n"
    "   \"edgemap\": {\n";

    for(int i=0;i<this->matrixSize;i++){
        if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
        else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
        else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
        else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
        else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
        for(int j=0;j<this->matrixSize;j++){
            if(this->edgeMatrix[i][j]=='"') std::cout « "\"\\\"\"";
            else if(this->edgeMatrix[i][j]=='\\') std::cout « "\"\\\\\"";
            else if(this->edgeMatrix[i][j]==0) std::cout « "\"\\\\x00\"";
            else if(this->edgeMatrix[i][j]<0) std::cout « "\"\\\\xff\"";
            else if(this->matrixIndex[i]=='\n') std::cout « "\"\\n\"";
            else std::cout « "\"" « this->edgeMatrix[i][j] « "\"";
            if(j!=this->matrixSize-1) std::cout « ", ";
        }
        std::cout « "],\n";
    }
    std::cout « "},\n";

    std::cout « "\"  weightmap\": {\n";
    for(int i=0;i<this->matrixSize;i++){
        if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
        else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
        else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
        else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
        else std::cout « "      \"" « this->matrixIndex[i] « "\": [";

        for(int j=0;j<this->matrixSize;j++){
            std::cout « this->valueMatrix[i][j];
            if(j!=this->matrixSize-1) std::cout « ", ";
        }
        std::cout « "],\n";
    }
    std::cout « "  }\n}\n";
}


void Markov::API::ModelMatrix::FastRandomWalkThread(std::mutex *mlock, std::ofstream *wordlist,
 unsigned long int n, int minLen, int maxLen, int id, bool bFileIO){
    if(n==0) return;

    Markov::Random::Marsaglia MarsagliaRandomEngine;
    char* e;
    char *res = new char[maxLen*n];
    int index = 0;
    char next;
    int len=0;
    long int selection;
    char cur;
    long int bufferctr = 0;
    for (int i = 0; i < n; i++) {
        cur=199;
        len=0;
        while (true) {
            e = strchr(this->matrixIndex, cur);
            index = e - this->matrixIndex;
            selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
            for(int j=0;j<this->matrixSize;j++){
                selection -= this->valueMatrix[index][j];
                if (selection < 0){
                    next = this->edgeMatrix[index][j];
                    break;
                }
            }

            if (len >= maxLen)  break;
            else if ((next < 0) && (len < minLen)) continue;
            else if (next < 0) break;
            cur = next;
            res[bufferctr + len++] = cur;
        }
        res[bufferctr + len++] = '\n';
        bufferctr+=len;

    }
    if(bFileIO){
        mlock->lock();
        *wordlist « res;
        mlock->unlock();
    }else{
        mlock->lock();
        std::cout « res;
        mlock->unlock();
    }
    delete res;

}


void Markov::API::ModelMatrix::FastRandomWalk(unsigned long int n, const char* wordlistFileName, int minLen, int maxLen, int threads, bool bFileIO){


    std::ofstream wordlist;
    if(bFileIO)
        wordlist.open(wordlistFileName);

    std::mutex mlock;
    if(n<=50000000ull) return this->FastRandomWalkPartition(&mlock, &wordlist, n, minLen, maxLen, bFileIO, threads);
    else{
        int numberOfPartitions = n/50000000ull;
        for(int i=0;i<numberOfPartitions;i++)
            this->FastRandomWalkPartition(&mlock, &wordlist, 50000000ull, minLen, maxLen, bFileIO, threads);
    }


}


void Markov::API::ModelMatrix::FastRandomWalkPartition(std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads){

    int iterationsPerThread = n/threads;
    int iterationsPerThreadCarryOver = n%threads;

    std::vector<std::thread*> threadsV;

    int id = 0;
    for(int i=0;i<threads;i++){
        threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
        id++;
    }

    threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock, wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));

    for(int i=0;i<threads;i++){
        threadsV[i]->join();
    }
}
```

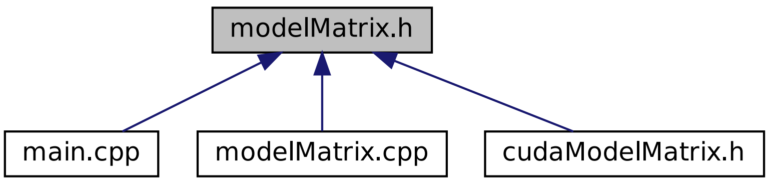
9.45 modelMatrix.h File Reference

```
#include "markovPasswords.h"
#include <mutex>
```

Include dependency graph for modelMatrix.h:



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



Classes

- class Markov::API::ModelMatrix

    Class to flatten and reduce Markov::Model to a Matrix.

Namespaces

- Markov

    Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

- Markov::API

    Namespace for the MarkovPasswords API.

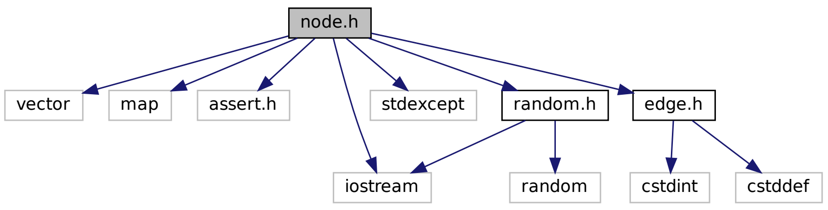
9.46 modelMatrix.h

```
#include "markovPasswords.h"
#include <mutex>

namespace Markov::API{

    /** @brief Class to flatten and reduce Markov::Model to a Matrix
     *
     * Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)
     *
     * To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are sychronized and files are flushed every 50M operations.
     *
    */
    class ModelMatrix : public Markov::API::MarkovPasswords{
    public:
        ModelMatrix();

        /** @brief Construct the related Matrix data for the model.
         *
         * This operation can be used after importing/training to allocate and populate the matrix content.
         *
         * this will initialize:
         * char** edgeMatrix -> a 2D array of mapping left and right connections of each edge.
         * long int **valueMatrix -> a 2D array representing the edge weights.
         * int matrixSize -> Size of the matrix, aka total number of nodes.
         * char* matrixIndex -> order of nodes in the model
         * long int *totalEdgeWeights -> total edge weights of each Node.
        */
        void ConstructMatrix();


        /** @brief Debug function to dump the model to a JSON file.
         *
         * Might not work 100%. Not meant for production use.
        */
        void DumpJSON();


        /** @brief Random walk on the Matrix-reduced Markov::Model
         *
         * This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
         *
         * If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations.
         * This comes at a minor performance penalty.
         *
         * While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
         *
         * This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.
         *
         * @param n - Number of passwords to generate.
         * @param wordlistFileName - Filename to write to
         * @param minLen - Minimum password length to generate
         * @param maxLen - Maximum password length to generate
         * @param threads - number of OS threads to spawn
         * @param bFileIO - If false, filename will be ignored and will output to stdout.
         *
         *
         * @code{.cpp}
         * Markov::API::ModelMatrix mp;
         * mp.Import("models/finished.mdl");
         * mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
         * @endcode
         *
        */
        void FastRandomWalk(unsigned long int n, const char* wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true);

    protected:

        /** @brief A single partition of FastRandomWalk event
         *
         * Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks,
         * large n parameters would lead to huge memory allocations.
         * @b Without @b Partitioning:
         * - 50M results 12 characters max -> 550 Mb Memory allocation
         *
         * - 5B results  12 characters max -> 55 Gb Memory allocation
         *
         * - 50B results 12 characters max -> 550GB Memory allocation
         *
         * Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.
         *
         * @param mlock - mutex lock to distribute to child threads
         * @param wordlist - Reference to the wordlist file to write to
         * @param n - Number of passwords to generate.
         * @param wordlistFileName - Filename to write to
         * @param minLen - Minimum password length to generate
         * @param maxLen - Maximum password length to generate
         * @param threads - number of OS threads to spawn
         * @param bFileIO - If false, filename will be ignored and will output to stdout.
         *
         *
        */
        void FastRandomWalkPartition(std::mutex *mlock, std::ofstream *wordlist, unsigned long int n,
      int minLen, int maxLen, bool bFileIO, int threads);

        /** @brief A single thread of a single partition of FastRandomWalk
         *
         * A FastRandomWalkPartition will initiate as many of this function as requested.
         *
         * This function contains the bulk of the generation algorithm.
         *
         * @param mlock - mutex lock to distribute to child threads
         * @param wordlist - Reference to the wordlist file to write to
         * @param n - Number of passwords to generate.
         * @param wordlistFileName - Filename to write to
         * @param minLen - Minimum password length to generate
         * @param maxLen - Maximum password length to generate
         * @param id - @b DEPRECATED Thread id - No longer used
         * @param bFileIO - If false, filename will be ignored and will output to stdout.
         *
         *
        */
        void FastRandomWalkThread(std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int
      minLen, int maxLen, int id, bool bFileIO);
        char** edgeMatrix;
        long int **valueMatrix;
        int matrixSize;
        char* matrixIndex;
        long int *totalEdgeWeights;
    };



};
```

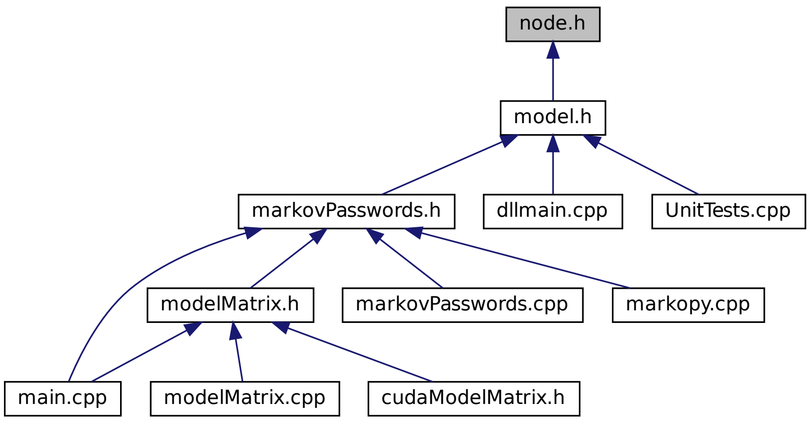
9.47 node.h File Reference

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

Include dependency graph for node.h:



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



Classes

- class Markov::Node< storageType >

  A node class that for the vertices of model. Connected with eachother using Edge.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

9.48 node.h

```
#pragma once
#include <vector>
#include <map>
#include <assert.h>
#include <iostream>
#include <stdexcept> // To use runtime_error
#include "edge.h"
#include "random.h"
namespace Markov {

    /** @brief 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*.
    */
    template <typename storageType>
    class Node {
    public:

        /** @brief Default constructor. Creates an empty Node.
        */
        Node<storageType>();

        /** @brief Constructor. Creates a Node with no edges and with given NodeValue.
         * @param _value - Nodes character representation.
         *
         * @b Example @b Use: Construct nodes
         * @code{.cpp}
         * Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
         * Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
         * @endcode
        */
        Node<storageType>(storageType _value);

        /** @brief Link this node with another, with this node as its source.
         *
         * Creates a new Edge.
         * @param target - Target node which will be the RightNode() of new edge.
         * @return A new node with LeftNode as this, and RightNode as parameter target.
         *
         * @b Example @b Use: Construct nodes
         * @code{.cpp}
         * 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);
         * @endcode
        */
        Edge<storageType>* Link(Node<storageType>*);

        /** @brief Link this node with another, with this node as its source.
         *
         * *DOES NOT* create a new Edge.
         * @param Edge - Edge that will accept this node as its LeftNode.
         * @return the same edge as parameter target.
         *
         * @b Example @b Use: Construct and link nodes
         * @code{.cpp}
         * 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);
         * @endcode
        */
        Edge<storageType>* Link(Edge<storageType>*);

        /** @brief 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.
         * @return Node that was chosen at EdgeWeight biased random.
         *
         * @b Example @b Use: Use randomNext to do a random walk on the model
         * @code{.cpp}
         *  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();
         *  }
         * @endcode
        */
        Node<storageType>* RandomNext(Markov::Random::RandomEngine* randomEngine);

        /** @brief Insert a new edge to the this.edges.
         * @param edge - New edge that will be inserted.
         * @return true if insertion was successful, false if it fails.
         *
         * @b Example @b Use: Construct and update edges
         *
         * @code{.cpp}
         * 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);
         * @endcode
        */
        bool UpdateEdges(Edge<storageType>*);

        /** @brief Find an edge with its character representation.
```

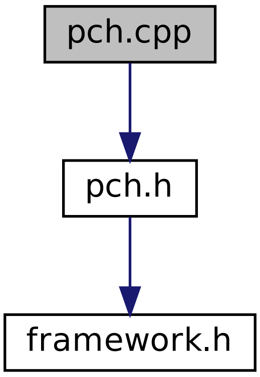
```
         * @param repr - character NodeValue of the target node.
         * @return Edge that is connected between this node, and the target node.
         *
         * @b Example @b Use: Construct and update edges
         *
         * @code{.cpp}
         * Markov::Node<unsigned char>* src = new 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');
         *
         * @endcode
         *
        */
        Edge<storageType>* FindEdge(storageType repr);

        /** @brief Find an edge with its pointer. Avoid unless neccessary because comptutational cost of find by character is cheaper (because of std::map)
        * @param target - target node.
        * @return Edge that is connected between this node, and the target node.
        */
        Edge<storageType>* FindEdge(Node<storageType>* target);

        /** @brief Return character representation of this node.
        * @return character representation at _value.
        */
        inline unsigned char NodeValue();

        /** @brief Change total weights with offset
         * @param offset to adjust the vertice weight with
        */
        void UpdateTotalVerticeWeight(long int offset);

        /** @brief return edges
        */
        inline std::map<storageType, Edge<storageType>*>* Edges();

        /** @brief return total edge weights
        */
        inline long int TotalEdgeWeights();


        std::vector<Edge<storageType>*> edgesV;
    private:


        storageType _value; /** @brief Character representation of this node. 0 for starter, 0xff for terminator.*/

        long int total_edge_weights;/** @brief Total weights of the vertices, required by RandomNext;*/

        /** @brief 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.
        */
        std::map<storageType, Edge<storageType>*> edges;
    };
};










template <typename storageType>
Markov::Node<storageType>::Node(storageType _value) {
    this->_value = _value;
    this->total_edge_weights = 0L;
};

template <typename storageType>
Markov::Node<storageType>::Node() {
    this->_value = 0;
    this->total_edge_weights = 0L;
};

template <typename storageType>
inline unsigned char Markov::Node<storageType>::NodeValue() {
    return _value;
}
```

```
template <typename storageType>
Markov::Edge<storageType>* Markov::Node<storageType>::Link(Markov::Node<storageType>* n) {
    Markov::Edge<storageType>* v = new Markov::Edge<storageType>(this, n);
    this->UpdateEdges(v);
    return v;
}

template <typename storageType>
Markov::Edge<storageType>* Markov::Node<storageType>::Link(Markov::Edge<storageType>* v) {
    v->SetLeftEdge(this);
    this->UpdateEdges(v);
    return v;
}

template <typename storageType>
Markov::Node<storageType>* Markov::Node<storageType>::RandomNext(Markov::Random::RandomEngine*
 randomEngine) {

    //get a random NodeValue in range of total_vertice_weight
    long int selection = randomEngine->random() %
 this->total_edge_weights;//distribution()(generator());// distribution(generator);
    //make absolute, no negative modulus values wanted
    //selection = (selection >= 0) ? selection : (selection + this->total_edge_weights);
    for(int i=0;i<this->edgesV.size();i++){
        selection -= this->edgesV[i]->EdgeWeight();
        if (selection < 0) return this->edgesV[i]->TraverseNode();
    }

    //if this assertion is reached, it means there is an implementation error above
    std::cout « "This should never be reached (node failed to walk to next)\n"; //cant assert from
 child thread
    assert(true && "This should never be reached (node failed to walk to next)");
    return NULL;
}

template <typename storageType>
bool Markov::Node<storageType>::UpdateEdges(Markov::Edge<storageType>* v) {
    this->edges.insert({ v->RightNode()->NodeValue(), v });
    this->edgesV.push_back(v);
    //this->total_edge_weights += v->EdgeWeight();
    return v->TraverseNode();
}

template <typename storageType>
Markov::Edge<storageType>* Markov::Node<storageType>::FindEdge(storageType repr) {
    auto e = this->edges.find(repr);
    if (e == this->edges.end()) return NULL;
    return e->second;
};

template <typename storageType>
void Markov::Node<storageType>::UpdateTotalVerticeWeight(long int offset) {
    this->total_edge_weights += offset;
}

template <typename storageType>
inline std::map<storageType, Markov::Edge<storageType>*>* Markov::Node<storageType>::Edges() {
    return &(this->edges);
}

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

9.49 pch.cpp File Reference

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

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



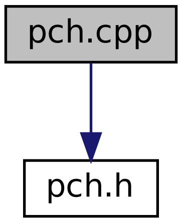
9.50 MarkovModel/src/pch.cpp

```
// pch.cpp: source file corresponding to the pre-compiled header

#include "pch.h"

// When you are using pre-compiled headers, this source file is necessary for compilation to succeed.
```

9.51 pch.cpp File Reference

`#include "pch.h"`

Include dependency graph for UnitTests/pch.cpp:



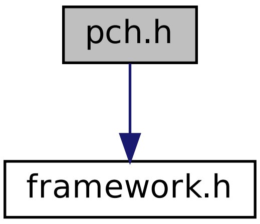
9.52 UnitTests/pch.cpp

```
// pch.cpp: source file corresponding to the pre-compiled header

#include "pch.h"

// When you are using pre-compiled headers, this source file is necessary for compilation to succeed.
```

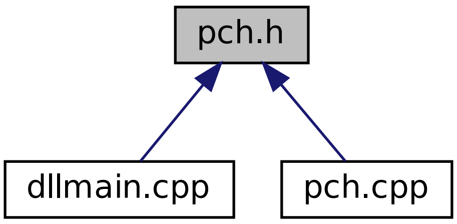
9.53 pch.h File Reference

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

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



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

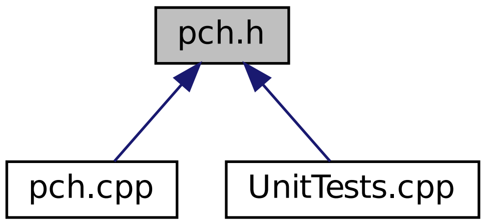


9.54 MarkovModel/src/pch.h

```
// pch.h: This is a precompiled header file.
// Files listed below are compiled only once, improving build performance for future builds.
// This also affects IntelliSense performance, including code completion and many code browsing
   features.
// However, files listed here are ALL re-compiled if any one of them is updated between builds.
// Do not add files here that you will be updating frequently as this negates the performance
   advantage.

#ifndef PCH_H
#define PCH_H

// add headers that you want to pre-compile here
#include "framework.h"

#endif //PCH_H
```

9.55 pch.h File Reference

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



9.56 UnitTests/pch.h

```
// pch.h: This is a precompiled header file.
// Files listed below are compiled only once, improving build performance for future builds.
// This also affects IntelliSense performance, including code completion and many code browsing
 features.
// However, files listed here are ALL re-compiled if any one of them is updated between builds.
// Do not add files here that you will be updating frequently as this negates the performance
 advantage.

#ifndef PCH_H
#define PCH_H

// add headers that you want to pre-compile here

#endif //PCH_H
```

9.57 random-model.py File Reference

Namespaces

- random-model
- random

Variables

- random-model.alphabet = string.printable

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

  output file handle

9.58 random-model.py

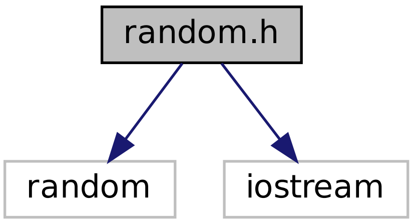
```
#!/usr/bin/python3
"""
  python script for generating a 2gram model
"""

import string
import re


alphabet = string.printable
alphabet = re.sub(’\s’, ”, alphabet)
print(f"alphabet={alphabet}")
```

```
#exit()


f = open('../../models/random.mdl', "wb")
#tie start nodes
for sym in alphabet:
    f.write(b"\x00,1," + bytes(sym, encoding='ascii') + b"\n")

#tie terminator nodes
for sym in alphabet:
    f.write(bytes(sym, encoding='ascii')+ b",1,\xff\n")

#tie internals
for src in alphabet:
    for target in alphabet:
        f.write(bytes(src, encoding='ascii') + b",1," + bytes(target, encoding='ascii') + b"\n")
```

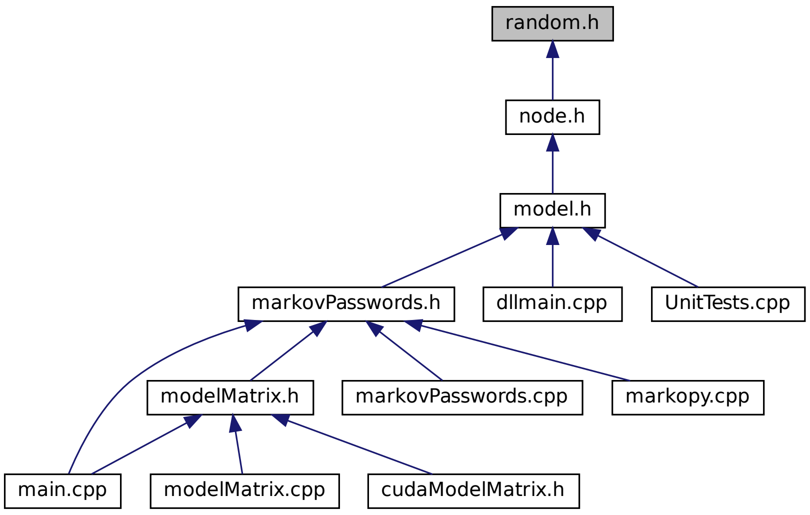
9.59 random.h File Reference

```
#include <random>
#include <iostream>
```

Include dependency graph for random.h:



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



Classes

- class Markov::Random::RandomEngine

    An abstract class for Random Engine.
- class Markov::Random::DefaultRandomEngine

    Implementation using Random.h default random engine.
- class Markov::Random::Marsaglia

    Implementation of Marsaglia Random Engine.
- class Markov::Random::Mersenne

    Implementation of Mersenne Twister Engine.

Namespaces

- Markov

    Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::Random

    Objects related to RNG.

9.60 random.h

```


#include <random>
#include <iostream>

/**
    @brief Objects related to RNG
*/
namespace Markov::Random{

    /** @brief 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.
     *
    */
    class RandomEngine{
    public:
        virtual inline unsigned long random() = 0;
    };



    /** @brief Implementation using Random.h default random engine
     *
     * This engine is also used by other engines for seeding.
     *
     *
     * @b Example @b Use: Using Default Engine with RandomWalk
     * @code{.cpp}
     * 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";
     *  }
     * @endcode
     *
     * @b Example @b Use: Generating a random number with Marsaglia Engine
     * @code{.cpp}
     * Markov::Random::DefaultRandomEngine de;
     * std::cout « de.random();
     * @endcode
     *
    */
    class DefaultRandomEngine : public RandomEngine{
```

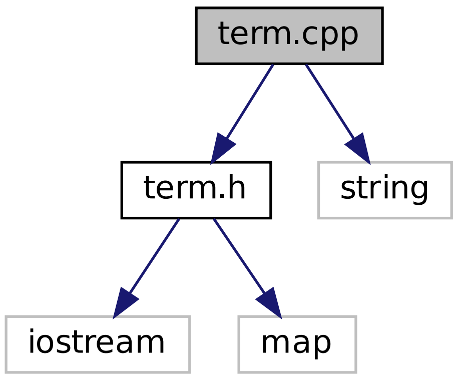
```
    public:
        /** @brief Generate Random Number
        * @return random number in long range.
        */
        inline unsigned long random(){
            return this->distribution()(this->generator());
        }
    protected:

        /** @brief Default random device for seeding
        *
        */
        inline std::random_device& rd() {
            static std::random_device _rd;
            return _rd;
        }

        /** @brief Default random engine for seeding
        *
        */
        inline std::default_random_engine& generator() {
            static std::default_random_engine _generator(rd()());
            return _generator;
        }

        /** @brief Distribution schema for seeding.
        *
        */
        inline std::uniform_int_distribution<long long unsigned>& distribution() {
            static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffFFFF);
            return _distribution;
        }

    };


    /** @brief 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.
     *
     * @b Example @b Use: Using Marsaglia Engine with RandomWalk
     * @code{.cpp}
     * 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";
     *  }
     * @endcode
     *
     * @b Example @b Use: Generating a random number with Marsaglia Engine
     * @code{.cpp}
     * Markov::Random::Marsaglia me;
     * std::cout « me.random();
     * @endcode
     *
    */
    class Marsaglia : public DefaultRandomEngine{
    public:

        /** @brief Construct Marsaglia Engine
        *
        * Initialize x,y and z using the default random engine.
        */
        Marsaglia(){
            this->x = this->distribution()(this->generator());
            this->y = this->distribution()(this->generator());
            this->z = this->distribution()(this->generator());
            //std::cout « "x: " « x « ", y: " « y « ", z: " « z « "\n";
        }


    inline unsigned long random(){
        unsigned long t;
        x ^= x « 16;
        x ^= x » 5;
        x ^= x « 1;

        t = x;
        x = y;
        y = z;
        z = t ^ x ^ y;

        return z;
    }


        unsigned long x;
        unsigned long y;
        unsigned long z;
    };


    /** @brief 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.
     *
     *
     * @b Example @b Use: Using Mersenne Engine with RandomWalk
     * @code{.cpp}
     * 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";
     *  }
     * @endcode
     *
     * @b Example @b Use: Generating a random number with Marsaglia Engine
     * @code{.cpp}
     * Markov::Random::Mersenne me;
     * std::cout « me.random();
     * @endcode
     *
    */
    class Mersenne : public DefaultRandomEngine{

    };


};
```

9.61 README.md File Reference

9.62 term.cpp File Reference

```
#include "term.h"
#include <string>
```

Include dependency graph for term.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const Terminal::color &c)

9.62.1 Function Documentation

9.62.1.1 operator<<()

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

overload for std::cout.

Definition at line 60 of file term.cpp.

```
                                                                       {
    char buf[6];
    sprintf(buf,"%d",Terminal::colormap.find(c)->second);
    os « "\e[1;" « buf « "m";
    return os;
}
```

References Markov::API::CLI::Terminal::colormap.

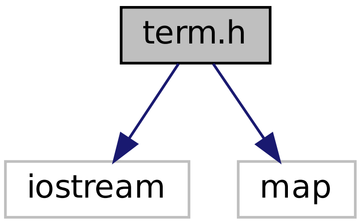
9.63 term.cpp

```
#pragma once
#include "term.h"
#include <string>

using namespace Markov::API::CLI;

//Windows text processing is different from unix systems, so use windows header and text attributes
#ifdef _WIN32

HANDLE Terminal::_stdout;
HANDLE Terminal::_stderr;

std::map<Terminal::color, DWORD> Terminal::colormap = {
    {Terminal::color::BLACK, 0},
    {Terminal::color::BLUE, 1},
    {Terminal::color::GREEN, 2},
    {Terminal::color::CYAN, 3},
    {Terminal::color::RED, 4},
    {Terminal::color::MAGENTA, 5},
    {Terminal::color::BROWN, 6},
    {Terminal::color::LIGHTGRAY, 7},
    {Terminal::color::DARKGRAY, 8},
    {Terminal::color::YELLOW, 14},
    {Terminal::color::WHITE, 15},
    {Terminal::color::RESET, 15},
};


Terminal::Terminal() {
    Terminal::_stdout = GetStdHandle(STD_OUTPUT_HANDLE);
    Terminal::_stderr = GetStdHandle(STD_ERROR_HANDLE);
}

std::ostream& operator«(std::ostream& os, const Terminal::color& c) {
    SetConsoleTextAttribute(Terminal::_stdout, Terminal::colormap.find(c)->second);
    return os;
}

#else

std::map<Terminal::color, int> Terminal::colormap = {
    {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},
};

Terminal::Terminal() {
    /*this->;*/
}

std::ostream& operator«(std::ostream& os, const Terminal::color& c) {
    char buf[6];
    sprintf(buf,"%d",Terminal::colormap.find(c)->second);
    os « "\e[1;" « buf « "m";
    return os;
}




#endif
```

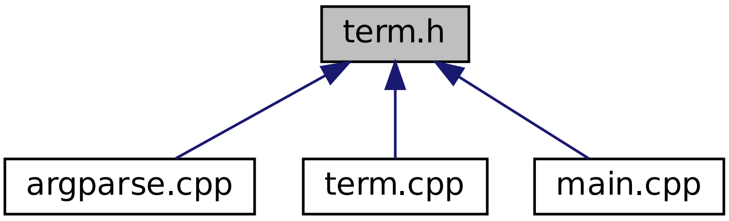
9.64 term.h File Reference

```
#include <iostream>
#include <map>
```

Include dependency graph for term.h:



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



Classes

- class Markov::API::CLI::Terminal

  pretty colors for Terminal. Windows Only.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.
- Markov::API::CLI

  Structure to hold parsed cli arguements.

Macros

- #define TERM_FAIL "[" << Markov::API::CLI::Terminal::color::RED << "+" << Markov::API::CLI::Terminal::color::RESET << "] "
- #define TERM_INFO "[" << Markov::API::CLI::Terminal::color::BLUE << "+" << Markov::API::CLI::Terminal::color::RESET << "] "
- #define TERM_WARN "[" << Markov::API::CLI::Terminal::color::YELLOW << "+" << Markov::API::CLI::Terminal::color::RESE << "] "
- #define TERM_SUCC "[" << Markov::API::CLI::Terminal::color::GREEN << "+" << Markov::API::CLI::Terminal::color::RESET << "] "

Functions

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

9.64.1 Macro Definition Documentation

9.64.1.1 TERM_FAIL

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

Definition at line 10 of file term.h.

9.64.1.2 TERM_INFO

```
#define TERM_INFO "[" << Markov::API::CLI::Terminal::color::BLUE << "+" << Markov::API::CLI::Terminal::colo
<< "] "
```
Definition at line 11 of file term.h.

9.64.1.3 TERM_SUCC

```
#define TERM_SUCC "[" << Markov::API::CLI::Terminal::color::GREEN << "+" << Markov::API::CLI::Terminal::col
<< "] "
```
Definition at line 13 of file term.h.

9.64.1.4 TERM_WARN

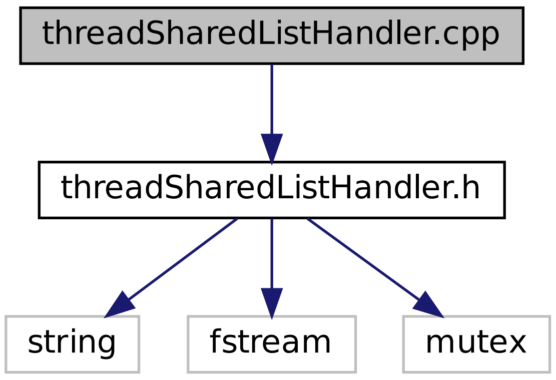
```
#define TERM_WARN "[" << Markov::API::CLI::Terminal::color::YELLOW << "+" << Markov::API::CLI::Terminal::co
<< "] "
```
Definition at line 12 of file term.h.

9.65 term.h

```
00001 #pragma once
00002
00003 #ifdef _WIN32
00004 #include <Windows.h>
00005 #endif
00006
00007 #include <iostream>
00008 #include <map>
00009
00010 #define TERM_FAIL "[" « Markov::API::CLI::Terminal::color::RED « "+" «
       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     /** @brief pretty colors for Terminal. Windows Only.
00017     */
00018     class Terminal {
00019     public:
00020
00021         /** Default constructor.
00022         * Get references to stdout and stderr handles.
00023         */
00024         Terminal();
00025
00026         enum color { RESET, BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, LIGHTGRAY,
       DARKGRAY, BROWN };
00027         #ifdef _WIN32
00028         static HANDLE _stdout;
00029         static HANDLE _stderr;
00030         static std::map<Markov::API::CLI::Terminal::color, DWORD> colormap;
00031         #else
00032         static std::map<Markov::API::CLI::Terminal::color, int> colormap;
00033         #endif
00034
00035
00036
00037         static std::ostream endl;
00038
00039
00040     };
00041
00042     /** overload for std::cout.
00043     */
00044     std::ostream& operator«(std::ostream& os, const Markov::API::CLI::Terminal::color& c);
00045
00046 }
```

9.66 threadSharedListHandler.cpp File Reference

`#include "threadSharedListHandler.h"`
Include dependency graph for threadSharedListHandler.cpp:



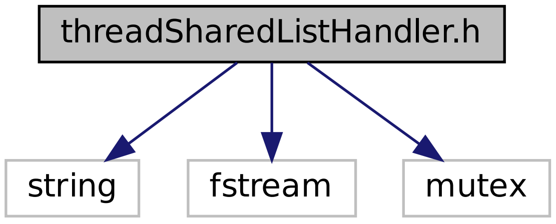
9.67 threadSharedListHandler.cpp

```
#include "threadSharedListHandler.h"


Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler(const char* filename){
    this->listfile;
    this->listfile.open(filename, std::ios_base::binary);
}


bool Markov::API::Concurrency::ThreadSharedListHandler::next(std::string* line){
    bool res = false;
    this->mlock.lock();
    res = (std::getline(this->listfile,*line,'\n'))? true : false;
    this->mlock.unlock();

    return res;
}
```

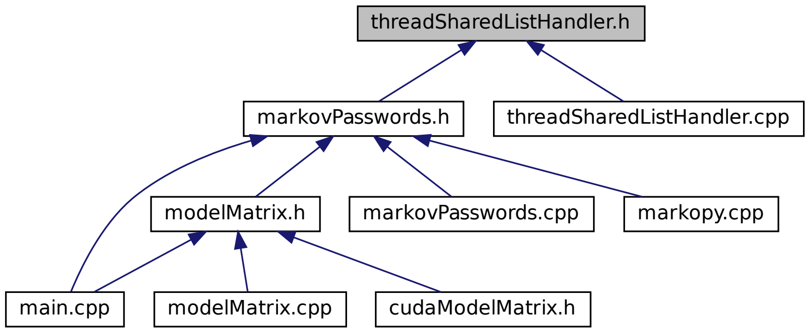
9.68 threadSharedListHandler.h File Reference

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

Include dependency graph for threadSharedListHandler.h:



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



Classes

- class Markov::API::Concurrency::ThreadSharedListHandler

  Simple class for managing shared access to file.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.
- Markov::API::Concurrency

  Namespace for Concurrency related classes.

9.69 threadSharedListHandler.h

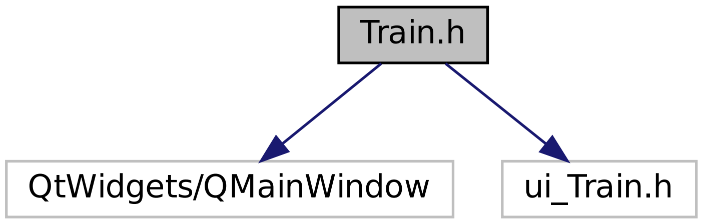
```
00001 #include <string>
00002 #include <fstream>
00003 #include <mutex>
00004
00005 /** @brief Namespace for Concurrency related classes
00006 */
00007 namespace Markov::API::Concurrency{
```

```

/** @brief 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.
 *
*/
class ThreadSharedListHandler{
public:
    /** @brief Construct the Thread Handler with a filename
     *
     * Simply open the file, and initialize the locks.
     *
     * @b Example @b Use: Simple file read
     * @code{.cpp}
     * ThreadSharedListHandler listhandler("test.txt");
     * std::string line;
     * std::cout « listhandler->next(&line) « "\n";
     * @endcode
     *
     * @b Example @b Use: Example use case from MarkovPasswords showing multithreaded access
     * @code{.cpp}
     *  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;
     *      }
     *   }
     * @endcode
     *
     * @param filename Filename for the file to manage.
    */
    ThreadSharedListHandler(const char* filename);

    /** @brief Read the next line from the file.
     *
     * This action will be blocked until another thread (if any) completes the read operation on the file.
     *
     * @b Example @b Use: Simple file read
     * @code{.cpp}
     * ThreadSharedListHandler listhandler("test.txt");
     * std::string line;
     * std::cout « listhandler->next(&line) « "\n";
     * @endcode
     *
    */
    bool next(std::string* line);

private:
    std::ifstream listfile;
    std::mutex mlock;
};

};
```

9.70 Train.h File Reference

```
#include <QtWidgets/QMainWindow>
#include "ui_Train.h"
```

Include dependency graph for Train.h:



Classes

- class Markov::GUI::Train

  QT Training page class.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

- Markov::GUI

  namespace for MarkovPasswords API GUI wrapper

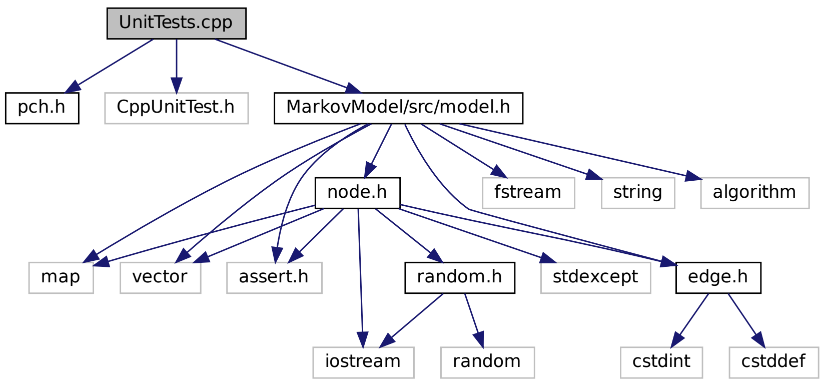
9.71 Train.h

```
#pragma once
#include <QtWidgets/QMainWindow>
#include "ui_Train.h"

namespace Markov::GUI{

    /** @brief QT Training page class
    */
    class Train :public QMainWindow {
    Q_OBJECT
    public:
        Train(QWidget* parent = Q_NULLPTR);

    private:
        Ui::Train ui;

    public slots:
        void home();
        void train();
    };
};
```

9.72 UnitTests.cpp File Reference

```
#include "pch.h"
#include "CppUnitTest.h"
#include "MarkovModel/src/model.h"
```

Include dependency graph for UnitTests.cpp:



Namespaces

- Testing

  Namespace for Microsoft Native Unit Testing Classes.
- Testing::MVP

  Testing Namespace for Minimal Viable Product.
- Testing::MVP::MarkovModel

  Testing Namespace for MVP MarkovModel.
- Testing::MVP::MarkovPasswords

  Testing namespace for MVP MarkovPasswords.
- Testing::MarkovModel

  Testing namespace for MarkovModel.
- Testing::MarkovPasswords

  Testing namespace for MarkovPasswords.

Functions

- Testing::MVP::MarkovModel::TEST_CLASS (Edge)

  Test class for minimal viable Edge.
- Testing::MVP::MarkovModel::TEST_CLASS (Node)

  Test class for minimal viable Node.
- Testing::MVP::MarkovModel::TEST_CLASS (Model)

  Test class for minimal viable Model.
- Testing::MVP::MarkovPasswords::TEST_CLASS (ArgParser)

  Test Class for Argparse class.
- Testing::MarkovModel::TEST_CLASS (Edge)

  Test class for rest of Edge cases.
- Testing::MarkovModel::TEST_CLASS (Node)

  Test class for rest of Node cases.
- Testing::MarkovModel::TEST_CLASS (Model)

  Test class for rest of model cases.

9.73 UnitTests.cpp

```
#include "pch.h"
#include "CppUnitTest.h"
#include "MarkovModel/src/model.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;


/** @brief Namespace for Microsoft Native Unit Testing Classes
*/
namespace Testing {

    /** @brief Testing Namespace for Minimal Viable Product
    */
    namespace MVP {
        /** @brief Testing Namespace for MVP MarkovModel
        */
        namespace MarkovModel
        {
            /** @brief Test class for minimal viable Edge
            */
            TEST_CLASS(Edge)
            {
            public:

                /** @brief test default constructor
                */
                TEST_METHOD(default_constructor) {
                    Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>;
                    Assert::IsNull(e->LeftNode());
                    Assert::IsNull(e->RightNode());
                    delete e;
                }

                /** @brief test linked constructor with two nodes
                */
                TEST_METHOD(linked_constructor) {
                    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 = new Markov::Edge<unsigned char>(LeftNode,
 RightNode);
                    Assert::IsTrue(LeftNode == e->LeftNode());
                    Assert::IsTrue(RightNode == e->RightNode());
                    delete LeftNode;
                    delete RightNode;
                    delete e;
                }

                /** @brief test AdjustEdge function
                */
                TEST_METHOD(AdjustEdge) {
                    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 = new Markov::Edge<unsigned char>(LeftNode,
 RightNode);
                    e->AdjustEdge(15);
                    Assert::AreEqual(15ull, e->EdgeWeight());
                    e->AdjustEdge(15);
                    Assert::AreEqual(30ull, e->EdgeWeight());
                    delete LeftNode;
                    delete RightNode;
                    delete e;
                }

                /** @brief test TraverseNode returning RightNode
                */
                TEST_METHOD(TraverseNode) {
                    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 = new Markov::Edge<unsigned char>(LeftNode,
 RightNode);
                    Assert::IsTrue(RightNode == e->TraverseNode());
                    delete LeftNode;
                    delete RightNode;
                    delete e;
                }

                /** @brief test LeftNode/RightNode setter
                */
                TEST_METHOD(set_left_and_right) {
                    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>* e1 = new Markov::Edge<unsigned char>(LeftNode,
 RightNode);

                    Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>;
                    e2->SetLeftEdge(LeftNode);
                    e2->SetRightEdge(RightNode);

                    Assert::IsTrue(e1->LeftNode() == e2->LeftNode());
                    Assert::IsTrue(e1->RightNode() == e2->RightNode());
                    delete LeftNode;
                    delete RightNode;
                    delete e1;
                    delete e2;
                }

                /** @brief test negative adjustments
                */
                TEST_METHOD(negative_adjust) {
                    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 = new Markov::Edge<unsigned char>(LeftNode,
       RightNode);
                    e->AdjustEdge(15);
                    Assert::AreEqual(15ull, e->EdgeWeight());
                    e->AdjustEdge(-15);
                    Assert::AreEqual(0ull, e->EdgeWeight());
                    delete LeftNode;
                    delete RightNode;
                    delete e;
                }
            };

            /** @brief Test class for minimal viable Node
            */
            TEST_CLASS(Node)
            {
            public:

                /** @brief test default constructor
                */
                TEST_METHOD(default_constructor) {
                    Markov::Node<unsigned char>* n = new Markov::Node<unsigned char>();
                    Assert::AreEqual((unsigned char)0, n->NodeValue());
                    delete n;
                }

                /** @brief test custom constructor with unsigned char
                */
                TEST_METHOD(uchar_constructor) {
                    Markov::Node<unsigned char>* n = NULL;
                    unsigned char test_cases[] = { 'c', 0x00, 0xff, -32 };
                    for (unsigned char tcase : test_cases) {
                        n = new Markov::Node<unsigned char>(tcase);
                        Assert::AreEqual(tcase, n->NodeValue());
                        delete n;
                    }
                }

                /** @brief test link function
                */
                TEST_METHOD(link_left) {
                    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);
                    delete LeftNode;
                    delete RightNode;
                    delete e;
                }

                /** @brief test link function
                */
                TEST_METHOD(link_right) {
                    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 = new Markov::Edge<unsigned char>(NULL, RightNode);
                    LeftNode->Link(e);
                    Assert::IsTrue(LeftNode == e->LeftNode());
                    Assert::IsTrue(RightNode == e->RightNode());
                    delete LeftNode;
                    delete RightNode;
                    delete e;
                }

                /** @brief test RandomNext with low values
                */
                TEST_METHOD(rand_next_low) {

                    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>* e = src->Link(target1);
                    e->AdjustEdge(15);
                    Markov::Node<unsigned char>* res = src->RandomNext();
                    Assert::IsTrue(res == target1);
                    delete src;
                    delete target1;
                    delete e;

                }

                /** @brief test RandomNext with 32 bit high values
                */
                TEST_METHOD(rand_next_u32) {

                    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>* e = src->Link(target1);
                    e->AdjustEdge(1 « 31);
                    Markov::Node<unsigned char>* res = src->RandomNext();
                    Assert::IsTrue(res == target1);
                    delete src;
                    delete target1;
                    delete e;

                }

                /** @brief random next on a node with no follow-ups
                */
                TEST_METHOD(rand_next_choice_1) {

                    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 = src->Link(target1);
                    Markov::Edge<unsigned char>* e2 = src->Link(target2);
                    e1->AdjustEdge(1);
                    e2->AdjustEdge((unsigned long)(1ull « 31));
                    Markov::Node<unsigned char>* res = src->RandomNext();
                    Assert::IsNotNull(res);
                    Assert::IsTrue(res == target2);
                    delete src;
                    delete target1;
                    delete e1;
                    delete e2;
                }

                /** @brief random next on a node with no follow-ups
                */
                TEST_METHOD(rand_next_choice_2) {

                    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 = src->Link(target1);
                    Markov::Edge<unsigned char>* e2 = src->Link(target2);
                    e2->AdjustEdge(1);
                    e1->AdjustEdge((unsigned long)(1ull « 31));
                    Markov::Node<unsigned char>* res = src->RandomNext();
                    Assert::IsNotNull(res);
                    Assert::IsTrue(res == target1);
                    delete src;
                    delete target1;
                    delete e1;
                    delete e2;
                }


                /** @brief test updateEdges
                */
                TEST_METHOD(update_edges_count) {

                    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);

                    Assert::AreEqual((size_t)2, src->Edges()->size());

                    delete src;
                    delete target1;
                    delete e1;
                    delete e2;
                }

                /** @brief test updateEdges
                */
                TEST_METHOD(update_edges_total) {

                    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);
                    Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target1);
                    e1->AdjustEdge(25);
                    src->UpdateEdges(e1);
                    e2->AdjustEdge(30);
                    src->UpdateEdges(e2);

                    Assert::AreEqual(55ull, src->TotalEdgeWeights());

                    delete src;
                    delete target1;
                    delete e1;
                    delete e2;
                }


                /** @brief test FindVertice
                */
                TEST_METHOD(find_vertice) {

                    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');
                    Assert::IsNotNull(res);
                    Assert::AreEqual((unsigned char)'b', res->TraverseNode()->NodeValue());
                    res = src->FindEdge('c');
                    Assert::IsNotNull(res);
                    Assert::AreEqual((unsigned char)'c', res->TraverseNode()->NodeValue());

                    delete src;
                    delete target1;
                    delete target2;


                }


                /** @brief test FindVertice
                */
                TEST_METHOD(find_vertice_without_any) {

                    auto _invalid_next = [] {
                        Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
                        Markov::Edge<unsigned char>* res = NULL;

                        res = src->FindEdge('b');
                        Assert::IsNull(res);

                        delete src;
                    };

                    //Assert::ExpectException<std::logic_error>(_invalid_next);
                }

                /** @brief test FindVertice
                */
                TEST_METHOD(find_vertice_nonexistent) {

                    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('D');
                    Assert::IsNull(res);

                    delete src;
                    delete target1;
                    delete target2;

                }
            };

            /** @brief Test class for minimal viable Model
            */
            TEST_CLASS(Model)
            {
            public:
                /** @brief test model constructor for starter node
                */
                TEST_METHOD(model_constructor) {
                    Markov::Model<unsigned char> m;
                    Assert::AreEqual((unsigned char)'\0', m.StarterNode()->NodeValue());
                }

                /** @brief test import
                */
                TEST_METHOD(import_filename) {
                    Markov::Model<unsigned char> m;
                    Assert::IsTrue(m.Import("../MarkovPasswords/Models/2gram.mdl"));
                }

                /** @brief test export
                */
                TEST_METHOD(export_filename) {
                    Markov::Model<unsigned char> m;
                    Assert::IsTrue(m.Export("../MarkovPasswords/Models/testcase.mdl"));
                }

                /** @brief test random walk
                */
                TEST_METHOD(random_walk) {
                    Markov::Model<unsigned char> m;
                    Assert::IsTrue(m.Import("../../models/finished.mdl"));
                    Assert::IsNotNull(m.RandomWalk(1,12));
                }
            };
        }

        /** @brief Testing namespace for MVP MarkovPasswords
        */
        namespace MarkovPasswords
        {
            /** @brief Test Class for Argparse class
            */
            TEST_CLASS(ArgParser)
            {
            public:
                /** @brief test basic generate
                */
                TEST_METHOD(generate_basic) {
                    int argc = 8;
                    char *argv[] = {"markov.exe", "generate", "-if", "model.mdl", "-of",
"passwords.txt", "-n", "100"};

                    /*ProgramOptions *p = Argparse::parse(argc, argv);
                    Assert::IsNotNull(p);

                    Assert::AreEqual(p->bImport, true);
                    Assert::AreEqual(p->bExport, false);
                    Assert::AreEqual(p->importname, "model.mdl");
                    Assert::AreEqual(p->outputfilename, "passwords.txt");
                    Assert::AreEqual(p->generateN, 100); */

                }

                /** @brief test basic generate reordered params
                */
                TEST_METHOD(generate_basic_reorder) {
                    int argc = 8;
                    char *argv[] = { "markov.exe", "generate", "-n", "100", "-if", "model.mdl", "-of",
"passwords.txt" };

                    /*ProgramOptions* p = Argparse::parse(argc, argv);
                    Assert::IsNotNull(p);

                    Assert::AreEqual(p->bImport, true);
                    Assert::AreEqual(p->bExport, false);
                    Assert::AreEqual(p->importname, "model.mdl");
                    Assert::AreEqual(p->outputfilename, "passwords.txt");
                    Assert::AreEqual(p->generateN, 100);*/
                }

                /** @brief test basic generate param longnames
                */
                TEST_METHOD(generate_basic_longname) {
                    int argc = 8;
                    char *argv[] = { "markov.exe", "generate", "-n", "100", "--inputfilename",
       "model.mdl", "--outputfilename", "passwords.txt" };

                    /*ProgramOptions* p = Argparse::parse(argc, argv);
                    Assert::IsNotNull(p);

                    Assert::AreEqual(p->bImport, true);
                    Assert::AreEqual(p->bExport, false);
                    Assert::AreEqual(p->importname, "model.mdl");
                    Assert::AreEqual(p->outputfilename, "passwords.txt");
                    Assert::AreEqual(p->generateN, 100); */
                }

                /** @brief test basic generate
                */
                TEST_METHOD(generate_fail_badmethod) {
                    int argc = 8;
                    char *argv[] = { "markov.exe", "junk", "-n", "100", "--inputfilename",
       "model.mdl", "--outputfilename", "passwords.txt" };

                    /*ProgramOptions* p = Argparse::parse(argc, argv);
                    Assert::IsNull(p); */
                }

                /** @brief test basic generate
                */
                TEST_METHOD(train_basic) {
                    int argc = 4;
                    char *argv[] = { "markov.exe", "train", "-ef", "model.mdl" };

                    /*ProgramOptions* p = Argparse::parse(argc, argv);
                    Assert::IsNotNull(p);

                    Assert::AreEqual(p->bImport, false);
                    Assert::AreEqual(p->bExport, true);
                    Assert::AreEqual(p->exportname, "model.mdl"); */

                }

                /** @brief test basic generate
                */
                TEST_METHOD(train_basic_longname) {
                    int argc = 4;
                    char *argv[] = { "markov.exe", "train", "--exportfilename", "model.mdl" };

                    /*ProgramOptions* p = Argparse::parse(argc, argv);
                    Assert::IsNotNull(p);

                    Assert::AreEqual(p->bImport, false);
                    Assert::AreEqual(p->bExport, true);
                    Assert::AreEqual(p->exportname, "model.mdl"); */
                }



            };

        }
    }


    /** @brief Testing namespace for MarkovModel
    */
    namespace MarkovModel {

        /** @brief Test class for rest of Edge cases
        */
        TEST_CLASS(Edge)
        {
        public:
            /** @brief send exception on integer underflow
            */
            TEST_METHOD(except_integer_underflow) {
                auto _underflow_adjust = [] {
                    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 = new Markov::Edge<unsigned char>(LeftNode,
       RightNode);
                    e->AdjustEdge(15);
                    e->AdjustEdge(-30);
                    delete LeftNode;
                    delete RightNode;
                    delete e;
                };
                Assert::ExpectException<std::underflow_error>(_underflow_adjust);
            }

            /** @brief test integer overflows
            */
            TEST_METHOD(except_integer_overflow) {
                auto _overflow_adjust = [] {
                    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 = new Markov::Edge<unsigned char>(LeftNode,
       RightNode);
                    e->AdjustEdge(~0ull);
                    e->AdjustEdge(1);
                    delete LeftNode;
                    delete RightNode;
                    delete e;
                };
                Assert::ExpectException<std::underflow_error>(_overflow_adjust);
            }
        };

        /** @brief Test class for rest of Node cases
        */
        TEST_CLASS(Node)
        {
        public:

            /** @brief test RandomNext with 64 bit high values
            */
            TEST_METHOD(rand_next_u64) {

                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>* e = src->Link(target1);
                e->AdjustEdge((unsigned long)(1ull « 63));
                Markov::Node<unsigned char>* res = src->RandomNext();
                Assert::IsTrue(res == target1);
                delete src;
                delete target1;
                delete e;

            }

            /** @brief test RandomNext with 64 bit high values
            */
            TEST_METHOD(rand_next_u64_max) {

                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>* e = src->Link(target1);
                e->AdjustEdge((0xffffFFFF));
                Markov::Node<unsigned char>* res = src->RandomNext();
                Assert::IsTrue(res == target1);
                delete src;
                delete target1;
                delete e;

            }

            /** @brief randomNext when no edges are present
            */
            TEST_METHOD(uninitialized_rand_next) {

                auto _invalid_next = [] {
                    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>* e = new Markov::Edge<unsigned char>(src, target1);
                    Markov::Node<unsigned char>* res = src->RandomNext();

                    delete src;
                    delete target1;
                    delete e;
                };

                Assert::ExpectException<std::logic_error>(_invalid_next);
            }


        };

        /** @brief Test class for rest of model cases
        */
        TEST_CLASS(Model)
        {
        public:
            TEST_METHOD(functional_random_walk) {
                Markov::Model<unsigned char> m;
                Markov::Node<unsigned char>* starter = m.StarterNode();
                Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
                Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
                Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
                Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
                starter->Link(a)->AdjustEdge(1);
                a->Link(b)->AdjustEdge(1);
                b->Link(c)->AdjustEdge(1);
                c->Link(end)->AdjustEdge(1);

                char* res = (char*)m.RandomWalk(1,12);
                Assert::IsFalse(strcmp(res, "abc"));
            }
            TEST_METHOD(functionoal_random_walk_without_any) {
                Markov::Model<unsigned char> m;
                Markov::Node<unsigned char>* starter = m.StarterNode();
                Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
                Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
                Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
                Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
                Markov::Edge<unsigned char>* res = NULL;
                starter->Link(a)->AdjustEdge(1);
                a->Link(b)->AdjustEdge(1);
                b->Link(c)->AdjustEdge(1);
                c->Link(end)->AdjustEdge(1);

                res = starter->FindEdge('D');
                Assert::IsNull(res);

            }
        };

    }

    /** @brief Testing namespace for MarkovPasswords
    */
    namespace MarkovPasswords {

    };

}
```

Index