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 PYTHON_PATH% (folder containing include, lib, libs, Tools, python.exe/python3.exe).
If you've set up the dependencies and environment variables correctly, you can open the solution with Visual Studio and build with that.

1.5 Known Common issues

1.5.1 Linux

1.5.1.1 Markopy - Python.h - Not found

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

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

Run `ls /usr/lib/x86_64-linux-gnu/ | grep boost` and check the shared object filenames. A common issue is that 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

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 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

6.1.1 Detailed Description

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

6.1.2 Function Documentation

6.1.2.1 cli_generate()

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

Definition at line 114 of file markopy_cli.py.

```
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))

```

6.1.2.2 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

```

6.1.2.3 cli_train()

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

Definition at line 82 of file markopy_cli.py.

```
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)


```

6.1.3 Variable Documentation

6.1.3.1 action

`markopy_cli.action`

Definition at line 49 of file markopy_cli.py.

6.1.3.2 args

`markopy_cli.args = parser.parse_args()`

Definition at line 58 of file markopy_cli.py.

6.1.3.3 bulk

`markopy_cli.bulk`

Definition at line 139 of file markopy_cli.py.

6.1.3.4 corpus_list

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

Definition at line 130 of file markopy_cli.py.

6.1.3.5 default

`markopy_cli.default`

Definition at line 41 of file markopy_cli.py.

6.1.3.6 help

`markopy_cli.help`

Definition at line 27 of file markopy_cli.py.

6.1.3.7 model

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

Definition at line 132 of file markopy_cli.py.

6.1.3.8 model_base

`markopy_cli.model_base = input`

Definition at line 153 of file markopy_cli.py.

6.1.3.9 model_extension

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

6.1.3.10 model_list

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

6.1.3.11 output

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

6.1.3.12 output_file_name

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

6.1.3.13 output_forced

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

6.1.3.14 parser

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

6.1.3.15 True

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

6.2 Markov Namespace Reference

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

Namespaces

- API

    Namespace for the MarkovPasswords API.
- GUI

    namespace for MarkovPasswords API GUI wrapper
- Markopy
- Random

    Objects related to RNG.

Classes

- class Edge

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

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

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

6.2.1 Detailed Description

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

6.3 Markov::API Namespace Reference

Namespace for the MarkovPasswords API.

Namespaces

- CLI

    Structure to hold parsed cli arguements.
- Concurrency

    Namespace for Concurrency related classes.

Classes

- class MarkovPasswords

    Markov::Model with char represented nodes.

6.3.1 Detailed Description

Namespace for the MarkovPasswords API.

6.4 Markov::API::CLI Namespace Reference

Structure to hold parsed cli 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)

6.4.1 Detailed Description

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

6.4.2 Typedef Documentation

6.4.2.1 ProgramOptions

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

6.4.3 Function Documentation

6.4.3.1 operator<<()

```
std::ostream& Markov::API::CLI::operator<< (
              std::ostream & os,
              const Markov::API::CLI::Terminal::color & c )
```
overload for std::cout.
Definition at line 60 of file term.cpp.
```
00060                                                              {
00061     char buf[6];
00062     sprintf(buf,"%d",Terminal::colormap.find(c)->second);
00063     os « "\e[1;" « buf « "m";
00064     return os;
00065 }
```
References Markov::API::CLI::Terminal::colormap.

6.5 Markov::API::Concurrency Namespace Reference

Namespace for Concurrency related classes.

Classes

- class ThreadSharedListHandler

  Simple class for managing shared access to file.

6.5.1 Detailed Description

Namespace for Concurrency related classes.

6.6 Markov::GUI Namespace Reference

namespace for MarkovPasswords API GUI wrapper

Classes

- class about

  QT Class for about page.
- class CLI

  QT CLI Class.
- class MarkovPasswordsGUI

  Reporting UI.

- class menu

  QT Menu class.
- class Train

  QT Training page class.

6.6.1 Detailed Description

namespace for MarkovPasswords API GUI wrapper

6.7 Markov::Markopy Namespace Reference

Functions

- BOOST_PYTHON_MODULE (markopy)

6.7.1 Function Documentation

6.7.1.1 BOOST_PYTHON_MODULE()

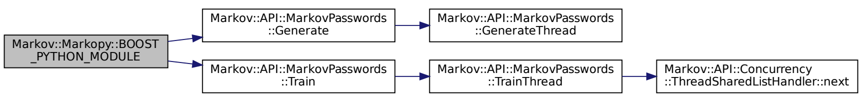
```
Markov::Markopy::BOOST_PYTHON_MODULE (
               markopy )
```

Definition at line 11 of file markopy.cpp.

```
00012    {
00013        bool  (Markov::API::MarkovPasswords::*Import)(const char*) = &Markov::Model<char>::Import;
00014        bool  (Markov::API::MarkovPasswords::*Export)(const char*) = &Markov::Model<char>::Export;
00015        class_<Markov::API::MarkovPasswords>("MarkovPasswords", init<>())
00016            .def(init<>())
00017            .def("Train", &Markov::API::MarkovPasswords::Train,
00018            "Train the model\n"
00019            "\n"
00020            ":param datasetFileName: Ifstream* to the dataset. If null, use class member\n"
00021            ":param delimiter: a character, same as the delimiter in dataset content\n"
00022            ":param threads: number of OS threads to spawn\n")
00023            .def("Generate", &Markov::API::MarkovPasswords::Generate,
00024            "Generate passwords from a trained model.\n"
00025            ":param n: Ifstream* to the dataset. If null, use class member\n"
00026            ":param wordlistFileName: a character, same as the delimiter in dataset content\n"
00027            ":param minLen: number of OS threads to spawn\n"
00028            ":param maxLen: Ifstream* to the dataset. If null, use class member\n"
00029            ":param threads: a character, same as the delimiter in dataset content\n"
00030            ":param threads: number of OS threads to spawn\n")
00031            .def("Import", Import, "Import a model file.")
00032            .def("Export", Export, "Export a model to file.")
00033        ;
00034    };
```

References Markov::API::MarkovPasswords::Generate(), and Markov::API::MarkovPasswords::Train().

Here is the call graph for this function:


Markov::Markopy::BOOST_PYTHON_MODULE
Markov::API::MarkovPasswords::Generate
Markov::API::MarkovPasswords::GenerateThread
Markov::API::MarkovPasswords::Train
Markov::API::MarkovPasswords::TrainThread
Markov::API::Concurrency::ThreadSharedListHandler::next

6.8 Markov::Random Namespace Reference

Objects related to RNG.

Classes

- class DefaultRandomEngine

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

  Implementation of Marsaglia Random Engine.
- class Mersenne

  Implementation of Mersenne Twister Engine.
- class RandomEngine

  An abstract class for Random Engine.

6.8.1 Detailed Description

Objects related to RNG.

6.9 model_2gram Namespace Reference

Variables

- alphabet = string.printable

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

  output file handle

6.9.1 Detailed Description

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

6.9.2 Variable Documentation

6.9.2.1 alphabet

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

6.9.2.2 f

`model_2gram.f = open('../../models/2gram.mdl', "wb")`
output file handle
Definition at line 16 of file model_2gram.py.
Referenced by Markov::Model< char >.Export(), and Markov::Model< char >.Import().

6.10 random Namespace Reference

6.10.1 Detailed Description

-model

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

6.11 random-model Namespace Reference

Variables

- alphabet = string.printable

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

  output file handle

6.11.1 Variable Documentation

6.11.1.1 alphabet

```
random-model.alphabet = string.printable
```

password alphabet
Definition at line 10 of file random-model.py.

6.11.1.2 f

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

output file handle
Definition at line 16 of file random-model.py.

6.12 Testing Namespace Reference

Namespace for Microsoft Native Unit Testing Classes.

Namespaces

- MarkovModel

  Testing namespace for MarkovModel.
- MarkovPasswords

  Testing namespace for MarkovPasswords.
- MVP

  Testing Namespace for Minimal Viable Product.

6.12.1 Detailed Description

Namespace for Microsoft Native Unit Testing Classes.

6.13 Testing::MarkovModel Namespace Reference

Testing namespace for MarkovModel.

Functions

- TEST_CLASS (Edge)

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

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

  Test class for rest of model cases.

6.13.1 Detailed Description

Testing namespace for MarkovModel.

6.13.2 Function Documentation

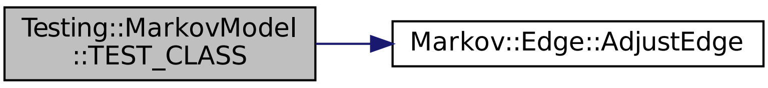
6.13.2.1 TEST_CLASS() [1/3]

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

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

```
{
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 Markov::Edge< NodeStorageType >::AdjustEdge().
Here is the call graph for this function:



6.13.2.2 TEST_CLASS() [2/3]

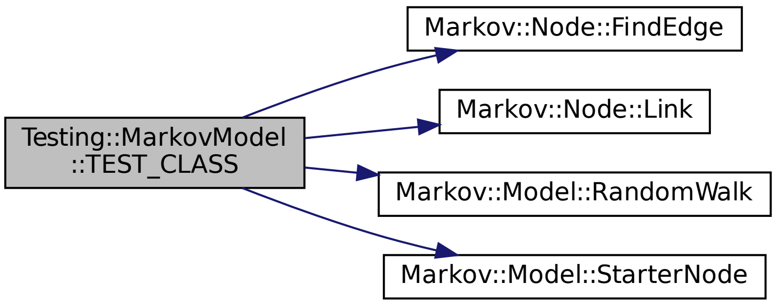
```
Testing::MarkovModel::TEST_CLASS (
              Model  )
```

Test class for rest of model cases.
Definition at line 589 of file UnitTests.cpp.

```
{
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 Markov::Node< storageType >::FindEdge(), Markov::Node< storageType >::Link(), Markov::Model< NodeStorageType
and Markov::Model< NodeStorageType >::StarterNode().

Here is the call graph for this function:


Markov::Node::FindEdge
Markov::Node::Link
Testing::MarkovModel ::TEST_CLASS
Markov::Model::RandomWalk
Markov::Model::StarterNode

6.13.2.3 TEST_CLASS() [3/3]

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

Test class for rest of Node cases.

test RandomNext with 64 bit high values

test RandomNext with 64 bit high values

randomNext when no edges are present

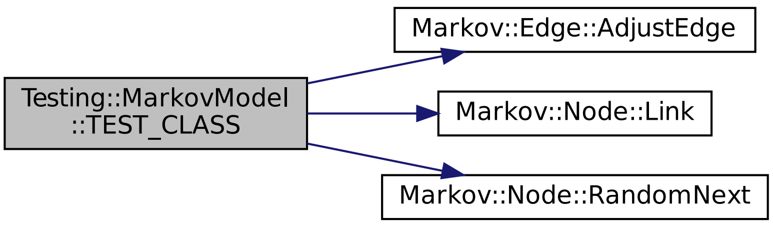
Definition at line 530 of file UnitTests.cpp.

```
    {
    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 Markov::Edge< NodeStorageType >::AdjustEdge(), Markov::Node< storageType >::Link(), and Markov::Node< storageType >::RandomNext().

Here is the call graph for this function:



6.14 Testing::MarkovPasswords Namespace Reference

Testing namespace for MarkovPasswords.

6.14.1 Detailed Description

Testing namespace for MarkovPasswords.

6.15 Testing::MVP Namespace Reference

Testing Namespace for Minimal Viable Product.

Namespaces

- MarkovModel

  Testing Namespace for MVP MarkovModel.
- MarkovPasswords

  Testing namespace for MVP MarkovPasswords.

6.15.1 Detailed Description

Testing Namespace for Minimal Viable Product.

6.16 Testing::MVP::MarkovModel Namespace Reference

Testing Namespace for MVP MarkovModel.

Functions

- TEST_CLASS (Edge)

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

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

  Test class for minimal viable Model.

6.16.1 Detailed Description

Testing Namespace for MVP MarkovModel.

6.16.2 Function Documentation

6.16.2.1 TEST_CLASS() [1/3]

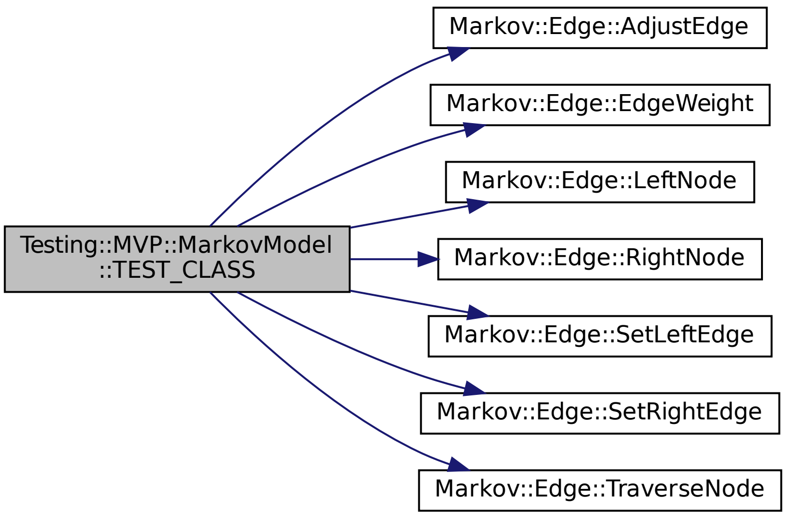
```
Testing::MVP::MarkovModel::TEST_CLASS (
              Edge  )
```

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

```
        {
        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 Markov::Edge< NodeStorageType >::AdjustEdge(), Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::Edge< NodeStorageType >::LeftNode(), Markov::Edge< NodeStorageType >::RightNode(), Markov::Edge< NodeStorage Markov::Edge< NodeStorageType >::SetRightEdge(), and Markov::Edge< NodeStorageType >::TraverseNode().

Here is the call graph for this function:


Testing::MVP::MarkovModel ::TEST_CLASS
Markov::Edge::AdjustEdge
Markov::Edge::EdgeWeight
Markov::Edge::LeftNode
Markov::Edge::RightNode
Markov::Edge::SetLeftEdge
Markov::Edge::SetRightEdge
Markov::Edge::TraverseNode

6.16.2.2 TEST_CLASS() [2/3]

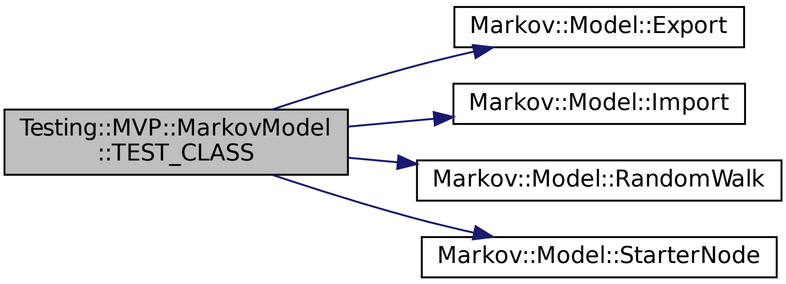
```
Testing::MVP::MarkovModel::TEST_CLASS (
                Model  )
```

Test class for minimal viable Model.
test model constructor for starter node
test import
test export
test random walk
Definition at line 347 of file UnitTests.cpp.

```
          {
          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 Markov::Model< NodeStorageType >::Export(), Markov::Model< NodeStorageType >::Import(), Markov::Model< NodeStorageType >::RandomWalk(), and Markov::Model< NodeStorageType >::StarterNode().

Here is the call graph for this function:


Testing::MVP::MarkovModel ::TEST_CLASS
Markov::Model::Export
Markov::Model::Import
Markov::Model::RandomWalk
Markov::Model::StarterNode

6.16.2.3 TEST_CLASS() [3/3]

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

Test class for minimal viable Node.
test default constructor
test custom constructor with unsigned char
test link function
test link function
test RandomNext with low values
test RandomNext with 32 bit high values
random next on a node with no follow-ups
random next on a node with no follow-ups
test updateEdges
test updateEdges
test FindVertice
test FindVertice
test FindVertice
Definition at line 111 of file UnitTests.cpp.

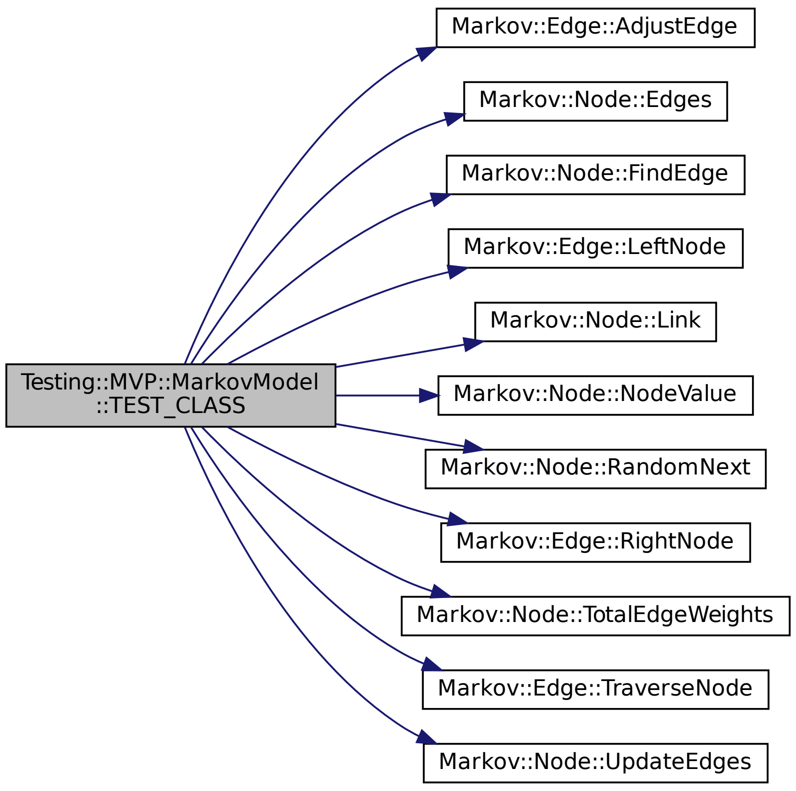
```
                  {
                  public:

                      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 Markov::Edge< NodeStorageType >::AdjustEdge(), Markov::Node< storageType >::Edges(), Markov::Node< storageType >::FindEdge(), Markov::Edge< NodeStorageType >::LeftNode(), Markov::Node< storageType >::Link
Markov::Node< storageType >::NodeValue(), Markov::Node< storageType >::RandomNext(), Markov::Edge< NodeStorageType >
Markov::Node< storageType >::TotalEdgeWeights(), Markov::Edge< NodeStorageType >::TraverseNode(), and Markov::Node< storageType >::UpdateEdges().

Here is the call graph for this function:



6.17 Testing::MVP::MarkovPasswords Namespace Reference

Testing namespace for MVP MarkovPasswords.

Functions

- TEST_CLASS (ArgParser)

    Test Class for Argparse class.

6.17.1 Detailed Description

Testing namespace for MVP MarkovPasswords.

6.17.2 Function Documentation

6.17.2.1 TEST_CLASS()

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

Test Class for Argparse class.
test basic generate
test basic generate reordered params
test basic generate param longnames
test basic generate
test basic generate
test basic generate
Definition at line 387 of file UnitTests.cpp.

```
        {
        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"); */
                }



            };
```

Chapter 7

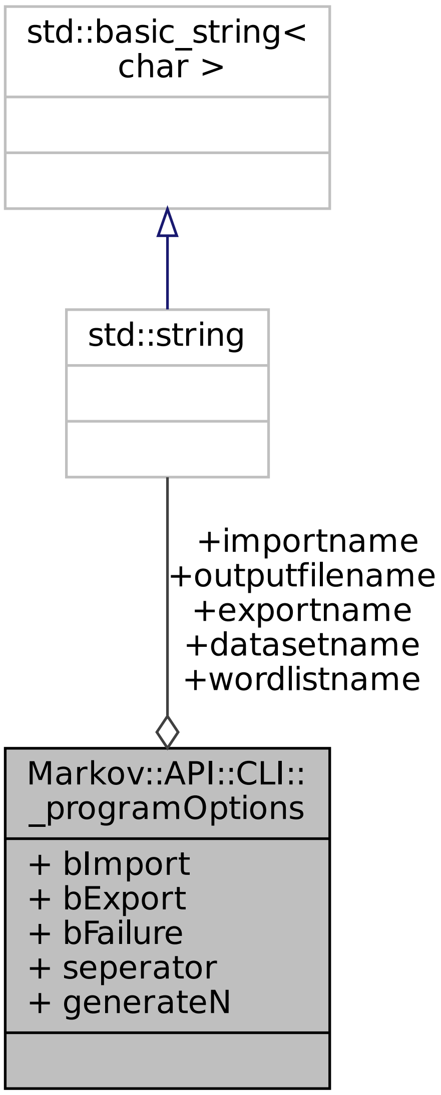
Class Documentation

7.1 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

7.1.1 Detailed Description

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

7.1.2 Member Data Documentation

7.1.2.1 bExport

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

7.1.2.2 bFailure

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

7.1.2.3 bImport

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

7.1.2.4 datasetname

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

7.1.2.5 exportname

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

7.1.2.6 generateN

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

7.1.2.7 importname

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

7.1.2.8 outputfilename

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

7.1.2.9 seperator

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

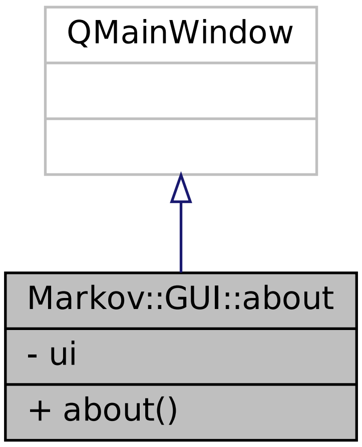
7.1.2.10 wordlistname

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

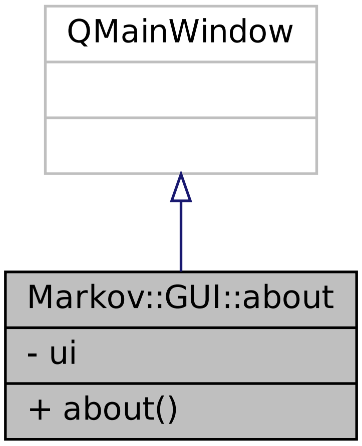
- argparse.h

7.2 Markov::GUI::about Class Reference

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



Collaboration diagram for Markov::GUI::about:



Public Member Functions

- about (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::main ui

7.2.1 Detailed Description

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

7.2.2 Constructor & Destructor Documentation

7.2.2.1 about()

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

7.2.3 Member Data Documentation

7.2.3.1 ui

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

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

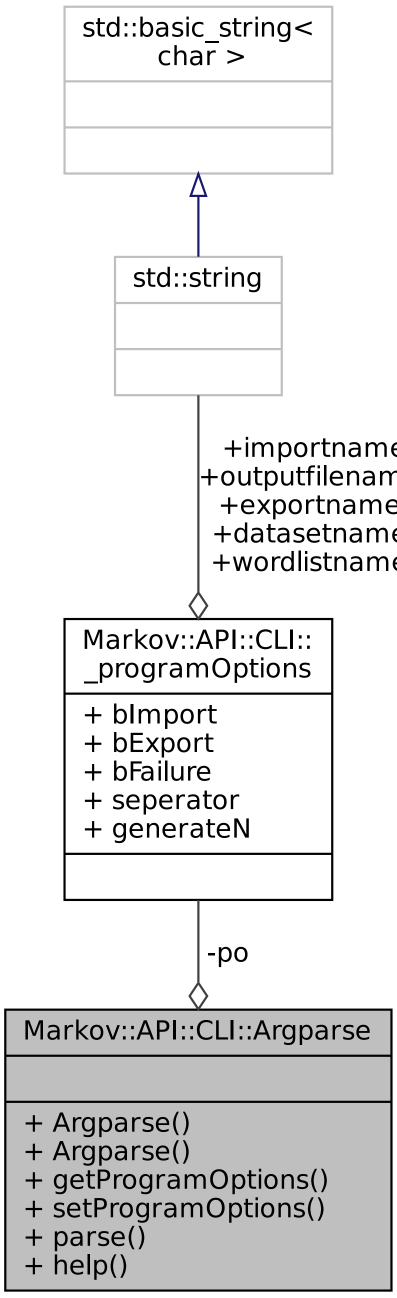
- about.h

7.3 Markov::API::CLI::Argparse Class Reference

Parse command line arguements.

`#include <argparse.h>`

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



Public Member Functions

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

Parse command line 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

7.3.1 Detailed Description

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

7.3.2 Constructor & Destructor Documentation

7.3.2.1 Argparse() [1/2]

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

7.3.2.2 Argparse() [2/2]

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

Parse command line 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.

```
                                           {

                 /*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;
    }
```

References Markov::API::CLI::_programOptions::bExport, Markov::API::CLI::_programOptions::bFailure, Markov::API::CLI::_programO
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.

7.3.3 Member Function Documentation

7.3.3.1 getProgramOptions()

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

Getter for command line options.

Getter for ProgramOptions populated by the arguement parser

Returns

ProgramOptions structure.

Definition at line 155 of file argparse.h.

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

References po.

7.3.3.2 help()

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

Print help string.

Definition at line 8 of file argparse.cpp.

```
                                   {
    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;
}
```

7.3.3.3 parse()

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

parse cli commands and return

Parameters

argc	- Program arguement count
argv	- Program arguement values array

Returns

ProgramOptions structure.

Definition at line 4 of file argparse.cpp.

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

7.3.3.4 setProgramOptions()

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

Initialize program options structure.

Parameters

i	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.

```
00172
                                          {
00173             this->po.bImport = i;
00174             this->po.bExport = e;
00175             this->po.seperator = s;
00176             this->po.bFailure = bf;
00177             this->po.generateN = n;
00178             this->po.importname = iName;
00179             this->po.exportname = exName;
00180             this->po.outputfilename = oName;
00181             this->po.datasetname = dName;
00182
00183             /*strcpy_s(this->po.importname,256,iName);
00184             strcpy_s(this->po.exportname,256,exName);
00185             strcpy_s(this->po.outputfilename,256,oName);
00186             strcpy_s(this->po.datasetname,256,dName);*/
00187
00188         }
```

References Markov::API::CLI::_programOptions::bExport, Markov::API::CLI::_programOptions::bFailure, Markov::API::CLI::_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.

7.3.4 Member Data Documentation

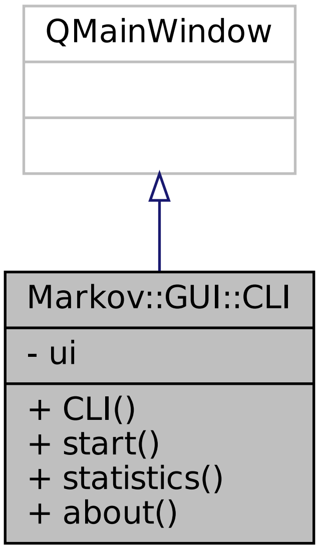
7.3.4.1 po

`Markov::API::CLI::ProgramOptions Markov::API::CLI::Argparse::po [private]`
Definition at line 203 of file argparse.h.
Referenced by Argparse(), getProgramOptions(), and setProgramOptions().
The documentation for this class was generated from the following files:

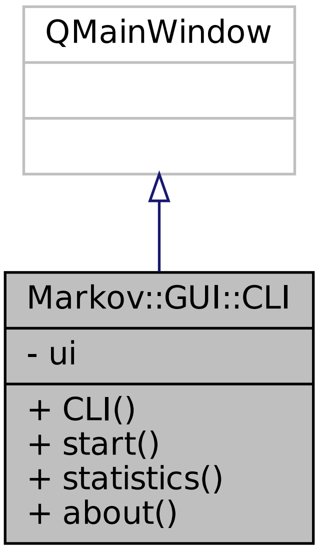
- argparse.h
- argparse.cpp

7.4 Markov::GUI::CLI Class Reference

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



Collaboration diagram for Markov::GUI::CLI:



Public Slots

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

Public Member Functions

- CLI (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::CLI ui

7.4.1 Detailed Description

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

7.4.2 Constructor & Destructor Documentation

7.4.2.1 CLI()

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

7.4.3 Member Function Documentation

7.4.3.1 about

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

7.4.3.2 start

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

7.4.3.3 statistics

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

7.4.4 Member Data Documentation

7.4.4.1 ui

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

Definition at line 14 of file CLI.h.

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

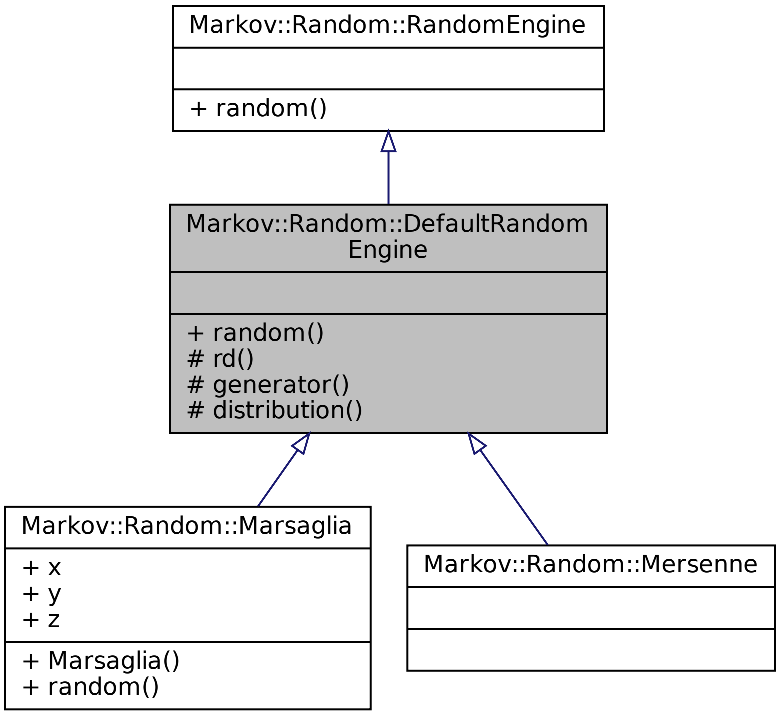
- CLI.h

7.5 Markov::Random::DefaultRandomEngine Class Reference

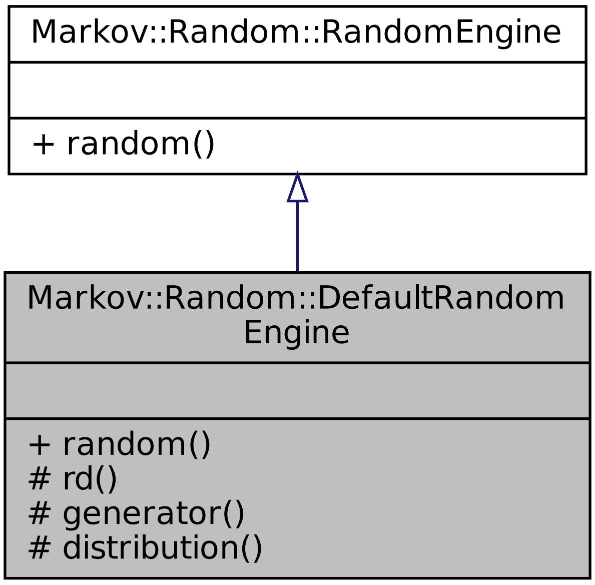
Implementation using Random.h default random engine.

`#include <random.h>`

Inheritance diagram for Markov::Random::DefaultRandomEngine:



Collaboration diagram for Markov::Random::DefaultRandomEngine:


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.

7.5.1 Detailed Description

Implementation using Random.h default random engine.
This engine is also used by other engines for seeding.
Example Use: Using Default Engine with RandomWalk

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

Example Use: Generating a random number with Marsaglia Engine

```
Markov::Random::DefaultRandomEngine de;
std::cout « de.random();
```

Definition at line 52 of file random.h.

7.5.2 Member Function Documentation

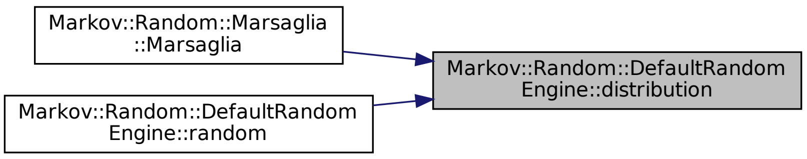
7.5.2.1 distribution()

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

Distribution schema for seeding.
Definition at line 81 of file random.h.

```
                                                                        {
               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

7.5.2.2 generator()

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

Default random engine for seeding.

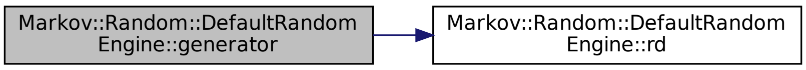
Definition at line 73 of file random.h.

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

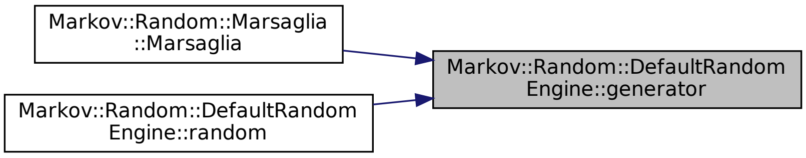
References rd().

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

7.5.2.3 random()

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

Generate Random Number.

Returns

random number in long range.

Implements Markov::Random::RandomEngine.

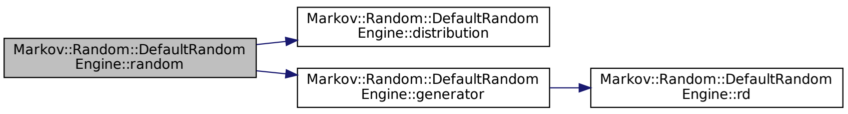
Reimplemented in Markov::Random::Marsaglia.

Definition at line 57 of file random.h.

```
                                          {
                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

7.5.2.4 rd()

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

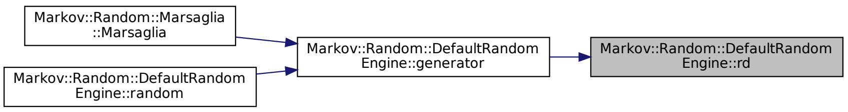
Default random device for seeding.

Definition at line 65 of file random.h.

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

Referenced by generator().

Here is the caller graph for this function:


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:

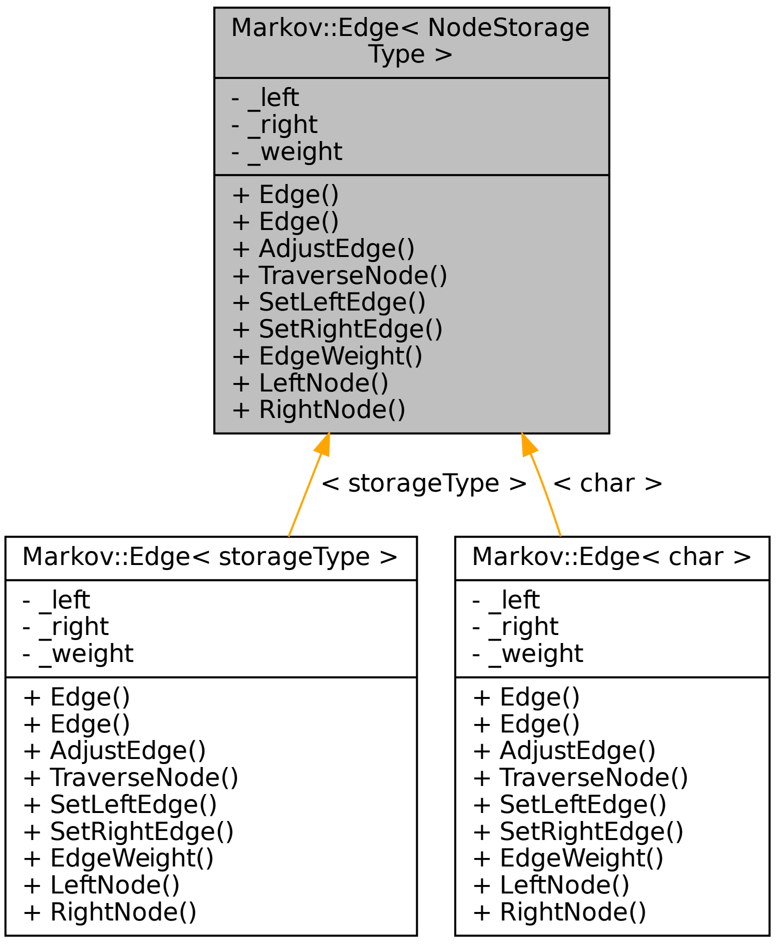
- random.h

7.6 Markov::Edge< NodeStorageType > Class Template Reference

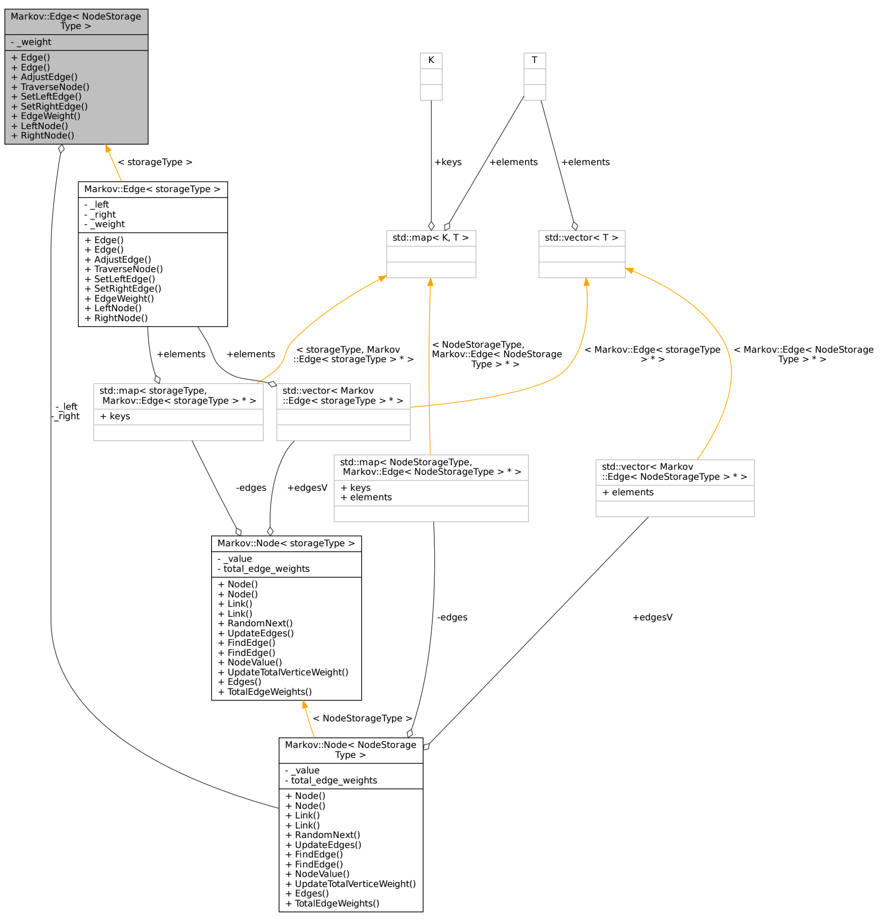
Edge class used to link nodes in the model together.

`#include <edge.h>`

Inheritance diagram for Markov::Edge< NodeStorageType >:



Collaboration diagram for Markov::Edge< NodeStorageType >:


Markov::Edge< NodeStorage Type >
- _weight
+ Edge()
+ Edge()
+ AdjustEdge()
+ TraverseNode()
+ SetLeftEdge()
+ SetRightEdge()
+ EdgeWeight()
+ LeftNode()
+ RightNode()
K
T
+keys
+elements
+elements
< storageType >
Markov::Edge< storageType >
- _left
- _right
- _weight
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

7.6.1 Detailed Description

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

Edge class used to link nodes in the model together.
Has LeftNode, RightNode, and EdgeWeight of the edge. Edges are *UNIDIRECTIONAL* in this model. They can only be traversed LeftNode to RightNode.
Definition at line 16 of file edge.h.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Edge() [1/2]

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

Default constructor.
Definition at line 105 of file edge.h.

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

7.6.2.2 Edge() [2/2]

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

Constructor. Initialize edge with given RightNode and LeftNode.

Parameters

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

Example Use: Construct edge

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

Definition at line 112 of file edge.h.

```

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

7.6.3 Member Function Documentation

7.6.3.1 AdjustEdge()

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

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

Parameters

offset	- NodeValue to be added to the EdgeWeight

Example Use: Construct edge

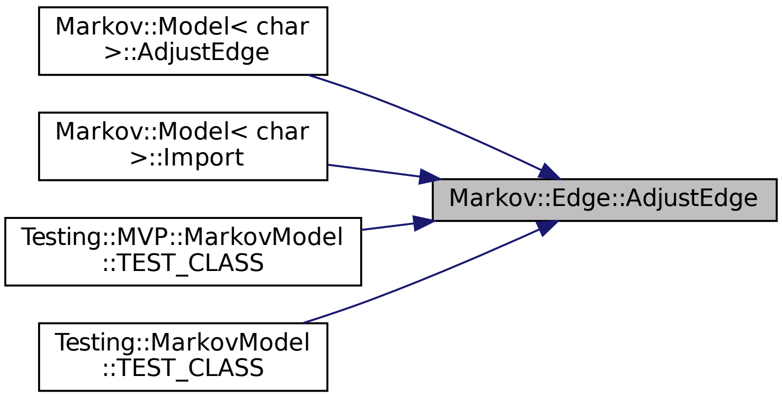
```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
e1->AdjustEdge(25);
```

Definition at line 119 of file edge.h.

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

Referenced by Markov::Model< char >::AdjustEdge(), Markov::Model< char >::Import(), Testing::MVP::MarkovModel::TEST_CLAS
and Testing::MarkovModel::TEST_CLASS().

Here is the caller graph for this function:


Markov::Model< char >::AdjustEdge
Markov::Model< char >::Import
Testing::MVP::MarkovModel ::TEST_CLASS
Testing::MarkovModel ::TEST_CLASS
Markov::Edge::AdjustEdge

7.6.3.2 EdgeWeight()

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

return edge's EdgeWeight.

Returns

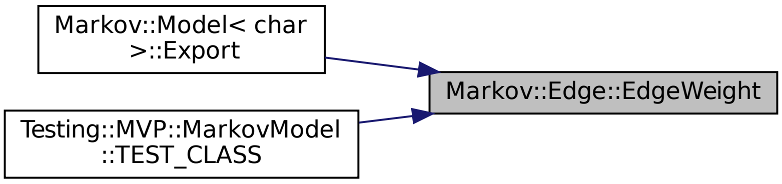
edge's EdgeWeight.

Definition at line 142 of file edge.h.

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

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

Here is the caller graph for this function:


Markov::Model< char >::Export
Testing::MVP::MarkovModel ::TEST_CLASS
Markov::Edge::EdgeWeight

7.6.3.3 LeftNode()

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

return edge's LeftNode

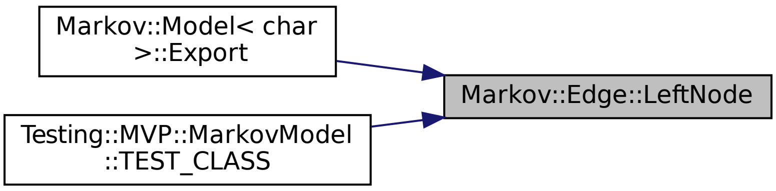
Returns

edge's LeftNode.

Definition at line 147 of file edge.h.

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

Referenced by Markov::Model< char >::Export(), and Testing::MVP::MarkovModel::TEST_CLASS().
Here is the caller graph for this function:


Markov::Model< char >::Export
Testing::MVP::MarkovModel ::TEST_CLASS
Markov::Edge::LeftNode

7.6.3.4 RightNode()

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

return edge's RightNode

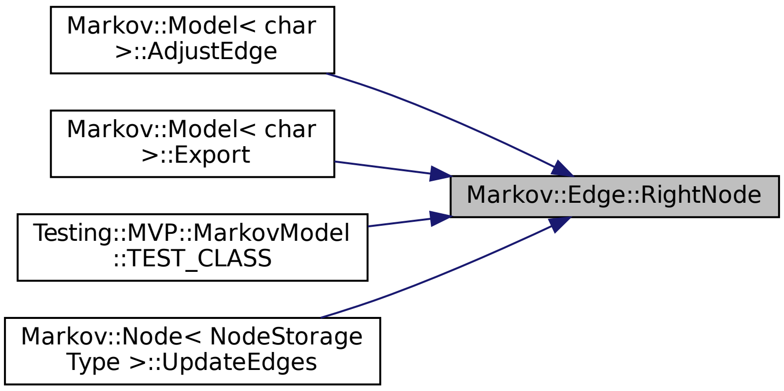
Returns

edge's RightNode.

Definition at line 152 of file edge.h.

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

Referenced by Markov::Model< char >::AdjustEdge(), Markov::Model< char >::Export(), Testing::MVP::MarkovModel::TEST_CLAS
and Markov::Node< NodeStorageType >::UpdateEdges().
Here is the caller graph for this function:


Markov::Model< char >::AdjustEdge
Markov::Model< char >::Export
Testing::MVP::MarkovModel ::TEST_CLASS
Markov::Node< NodeStorage Type >::UpdateEdges
Markov::Edge::RightNode

7.6.3.5 SetLeftEdge()

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

Set LeftNode of this edge.

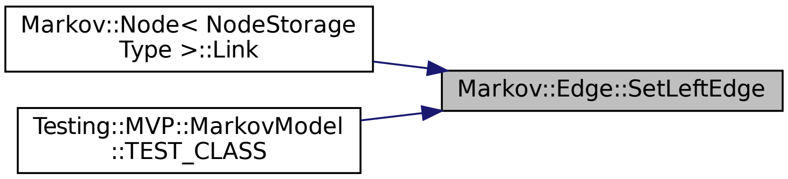
Parameters

node	- Node to be linked with.

Definition at line 132 of file edge.h.

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

Referenced by Markov::Node< NodeStorageType >::Link(), and Testing::MVP::MarkovModel::TEST_CLASS().
Here is the caller graph for this function:


Markov::Node< NodeStorage Type >::Link
Testing::MVP::MarkovModel ::TEST_CLASS
Markov::Edge::SetLeftEdge

7.6.3.6 SetRightEdge()

```
template<typename NodeStorageType >
```

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

Set RightNode of this edge.

Parameters

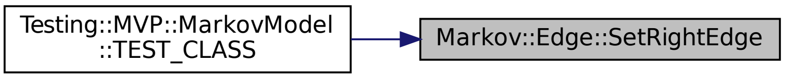
node	- Node to be linked with.

Definition at line 137 of file edge.h.

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

Referenced by Testing::MVP::MarkovModel::TEST_CLASS().

Here is the caller graph for this function:


Testing::MVP::MarkovModel ::TEST_CLASS
Markov::Edge::SetRightEdge

7.6.3.7 TraverseNode()

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

Traverse this edge to RightNode.

Returns

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

Example Use: Traverse a node

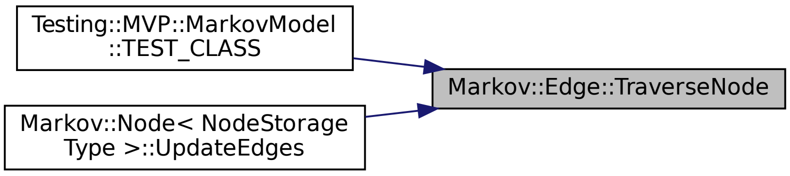
```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
e1->AdjustEdge(25);
Markov::Edge<unsigned char>* e2 = e1->traverseNode();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

Definition at line 125 of file edge.h.

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

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

Here is the caller graph for this function:


Testing::MVP::MarkovModel ::TEST_CLASS
Markov::Node< NodeStorage Type >::UpdateEdges
Markov::Edge::TraverseNode

7.6.4 Member Data Documentation

7.6.4.1 _left

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

Definition at line 95 of file edge.h.

7.6.4.2 _right

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

source node
Definition at line 96 of file edge.h.

7.6.4.3 _weight

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

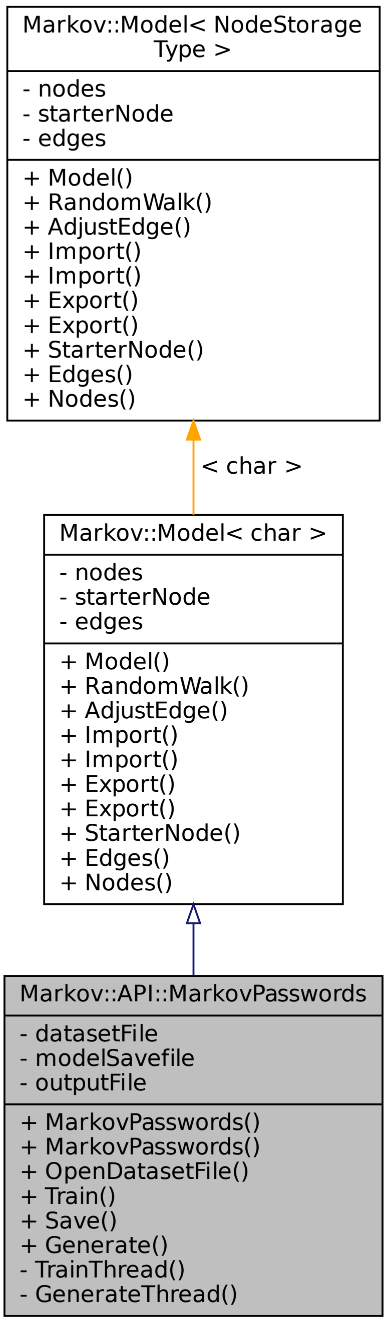
target node
Definition at line 97 of file edge.h.
The documentation for this class was generated from the following file:

- edge.h

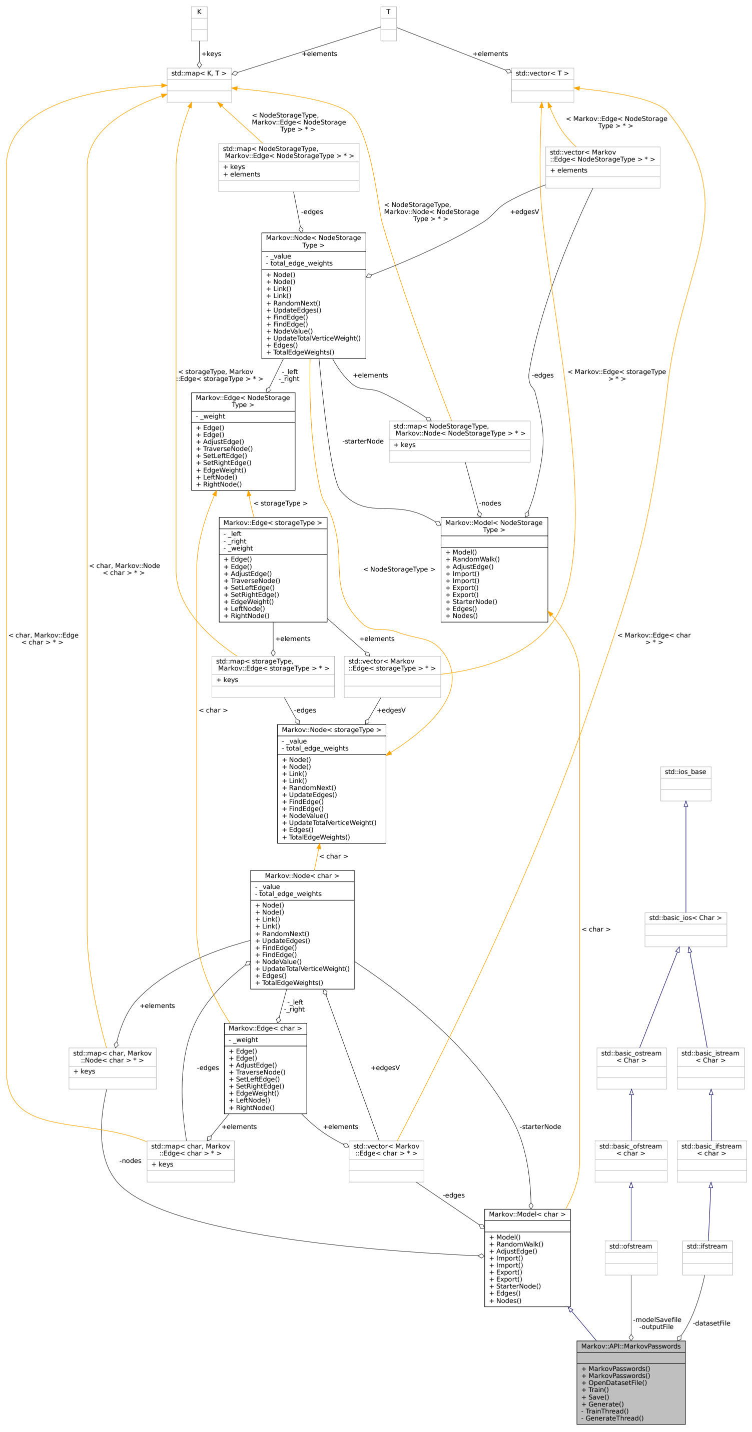
7.7 Markov::API::MarkovPasswords Class Reference

Markov::Model with char represented nodes.
`#include <markovPasswords.h>`

Inheritance diagram for Markov::API::MarkovPasswords:



Collaboration diagram for Markov::API::MarkovPasswords:



Public Member Functions

- [MarkovPasswords](#) ()

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

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

  Open dataset file and return the ifstream pointer.

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

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

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

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

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

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

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

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

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

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

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

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

    Return starter Node.

Private Member Functions

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

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

    A single thread invoked by the Generate function.

Private Attributes

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

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

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

    A list of all edges in this model.

7.7.1 Detailed Description

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

7.7.2 Constructor & Destructor Documentation

7.7.2.1 MarkovPasswords() [1/2]

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

Initialize the markov model from MarkovModel::Markov::Model.
Parent constructor. Has no extra functionality.
Definition at line 10 of file markovPasswords.cpp.

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


}
```

7.7.2.2 MarkovPasswords() [2/2]

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

Initialize the markov model from MarkovModel::Markov::Model, with an import file.
This function calls the Markov::Model::Import on the filename to construct the model. Same thing as creating and empty model, and calling MarkovPasswords::Import on the filename.

Parameters

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;

}
```

7.7.3 Member Function Documentation

7.7.3.1 AdjustEdge()

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

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

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

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with

Parameters

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;
}
```

7.7.3.2 Edges()

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

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 172 of file model.h.

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

7.7.3.3 Export() [1/2]

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

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

Returns

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

Example Use: Export file to filename

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

Definition at line 285 of file model.h.

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

7.7.3.4 Export() [2/2]

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

Export a file of the model.

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

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

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

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

Definition at line 273 of file model.h.

```
                                                                {
    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;
}
```

7.7.3.5 Generate()

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

Call Markov::Model::RandomWalk n times, and collect output.

Write the data to this->outputfile

Parameters

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

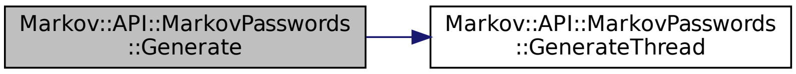
Definition at line 92 of file markovPasswords.cpp.

```
                               {
    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);

}
```

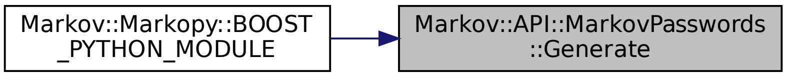
References GenerateThread().

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

Here is the call graph for this function:


Markov::API::MarkovPasswords ::Generate
Markov::API::MarkovPasswords ::GenerateThread

Here is the caller graph for this function:


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

7.7.3.6 GenerateThread()

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

A single thread invoked by the Generate function.

Parameters

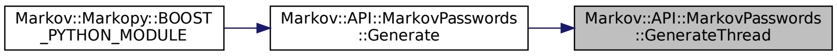
outputLock	- shared mutex lock to lock during output operation. Prevents race condition on write.
n	number of lines to be generated by this thread
wordlist	wordlistfile
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

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();
    }
}
```

Referenced by Generate().

Here is the caller graph for this function:


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

7.7.3.7 Import() [1/2]

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

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

Returns

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

Example Use: Import a file with filename

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

Definition at line 265 of file model.h.

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

}
```

7.7.3.8 Import() [2/2]

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

Import a file to construct the model.

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

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

Returns

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

Example Use: Import a file from ifstream

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

Definition at line 206 of file model.h.

```
                                                        {
    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";
```

```
00227         }
00228         else {
00229             srcN = this->nodes.find(src)->second;
00230         }
00231 
00232         if (this->nodes.find(target) == this->nodes.end()) {
00233             targetN = new Markov::Node<NodeStorageType>(target);
00234             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00235             //std::cout « "Creating new node at end.\n";
00236         }
00237         else {
00238             targetN = this->nodes.find(target)->second;
00239         }
00240         e = srcN->Link(targetN);
00241         e->AdjustEdge(oc);
00242         this->edges.push_back(e);
00243 
00244         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
00245 
00246 
00247     }
00248 
00249     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00250         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00251         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00252             return lhs->EdgeWeight() > rhs->EdgeWeight();
00253         });
00254         //for(int i=0;i<x.second->edgesV.size();i++)
00255         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00256         //std::cout « "\n";
00257     }
00258     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00259     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00260 
00261     return true;
00262 }
```

7.7.3.9 Nodes()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 177 of file model.h.

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

7.7.3.10 OpenDatasetFile()

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

Open dataset file and return the ifstream pointer.

Parameters

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;
```

```

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

7.7.3.11 RandomWalk()

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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
}
```

7.7.3.12 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;
}
```

7.7.3.13 StarterNode()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 167 of file model.h.

```
{ return starterNode;}
```

7.7.3.14 Train()

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

Train the model with the dataset file.

Parameters

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

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

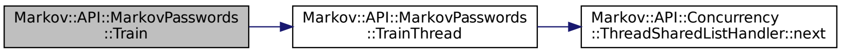
Definition at line 40 of file markovPasswords.cpp.

```
                                                                                                        {
    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";

}
```

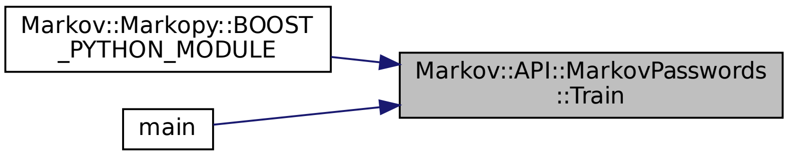
References TrainThread().

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

Here is the call graph for this function:


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

Here is the caller graph for this function:


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

7.7.3.15 TrainThread()

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

A single thread invoked by the Train function.

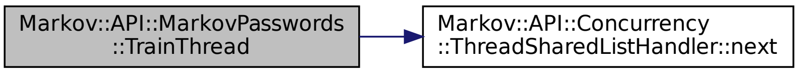
Parameters

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

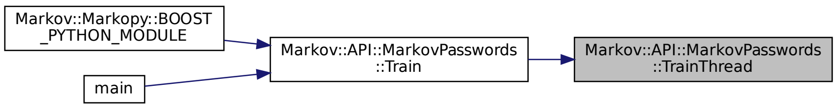
Definition at line 59 of file markovPasswords.cpp.

```
                {
    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::API::Concurrency::ThreadSharedListHandler::next().
Referenced by Train().
Here is the call graph for this function:


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

Here is the caller graph for this function:


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

7.7.4 Member Data Documentation

7.7.4.1 datasetFile

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

Definition at line 100 of file markovPasswords.h.

7.7.4.2 edges

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

A list of all edges in this model.
Definition at line 194 of file model.h.

7.7.4.3 modelSavefile

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

Definition at line 101 of file markovPasswords.h.

7.7.4.4 nodes

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

Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 183 of file model.h.

7.7.4.5 outputFile

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

Definition at line 102 of file markovPasswords.h.

7.7.4.6 starterNode

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

Starter Node of this model.

Definition at line 188 of file model.h.

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

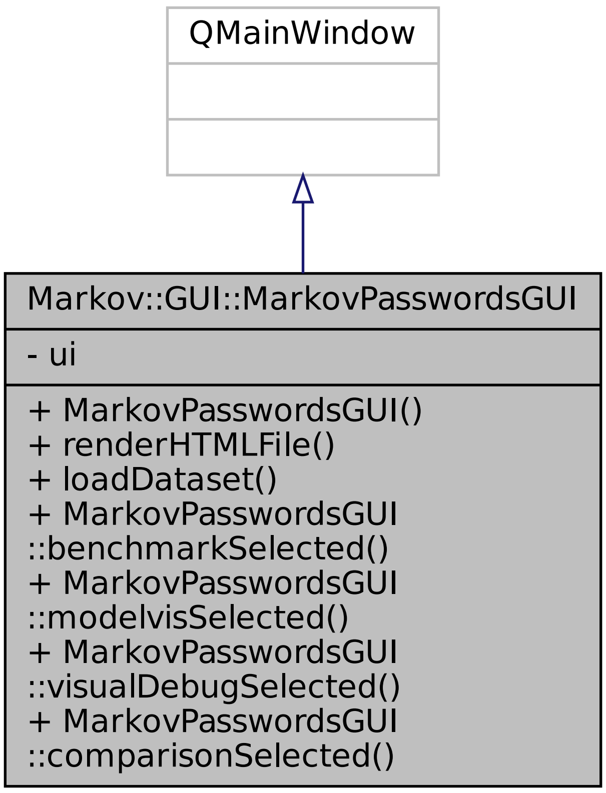
- markovPasswords.h
- markovPasswords.cpp

7.8 Markov::GUI::MarkovPasswordsGUI Class Reference

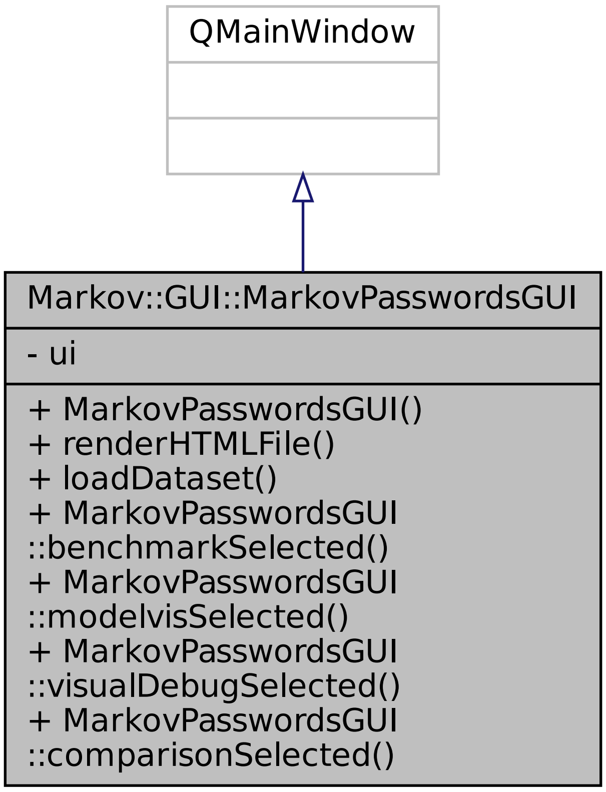
Reporting UI.

`#include <MarkovPasswordsGUI.h>`

Inheritance diagram for Markov::GUI::MarkovPasswordsGUI:


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

Collaboration diagram for Markov::GUI::MarkovPasswordsGUI:


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

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

7.8.1 Detailed Description

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

7.8.2 Constructor & Destructor Documentation

7.8.2.1 MarkovPasswordsGUI()

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

Default QT 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(); });
}
```

References ui.

7.8.3 Member Function Documentation

7.8.3.1 loadDataset()

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

Load a dataset to current view..

Parameters

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

}
```

7.8.3.2 MarkovPasswordsGUI::benchmarkSelected

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

7.8.3.3 MarkovPasswordsGUI::comparisonSelected

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

7.8.3.4 MarkovPasswordsGUI::modelvisSelected

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

7.8.3.5 MarkovPasswordsGUI::visualDebugSelected

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

7.8.3.6 renderHTMLFile()

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

Render a HTML file.

Parameters

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

Definition at line 71 of file MarkovPasswordsGUI.cpp.

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

7.8.4 Member Data Documentation

7.8.4.1 ui

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

Definition at line 32 of file MarkovPasswordsGUI.h.

Referenced by MarkovPasswordsGUI().

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

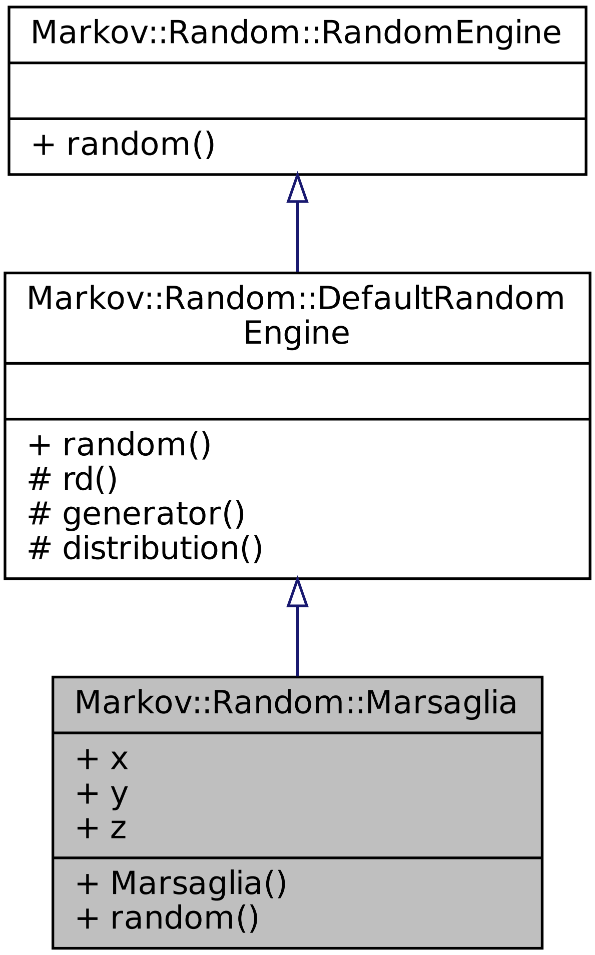
- MarkovPasswordsGUI.h
- MarkovPasswordsGUI.cpp

7.9 Markov::Random::Marsaglia Class Reference

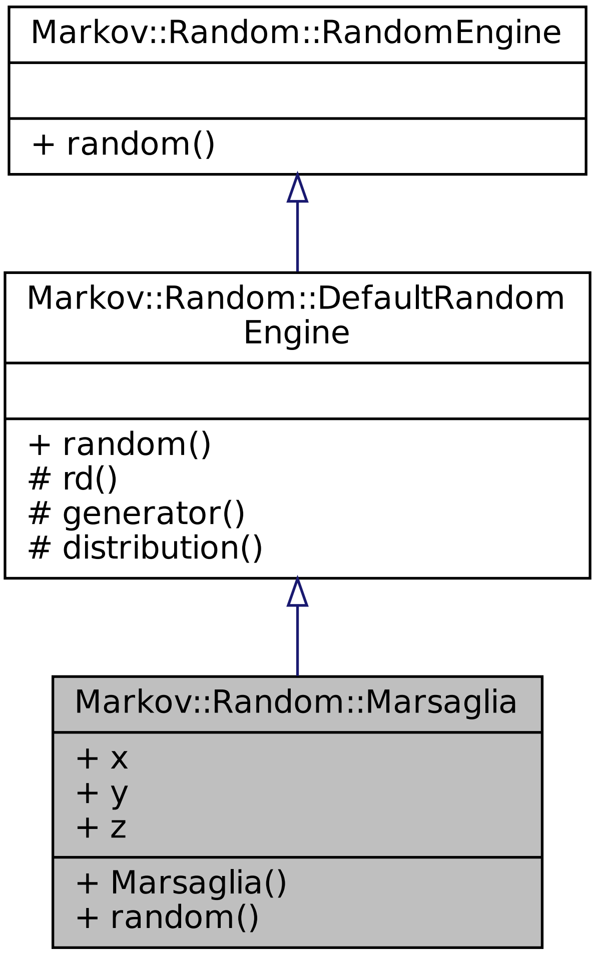
Implementation of Marsaglia Random Engine.

`#include <random.h>`

Inheritance diagram for Markov::Random::Marsaglia:


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.

7.9.1 Detailed Description

Implementation of Marsaglia Random Engine.
This is an implementation of Marsaglia Random engine, which for most use cases is a better fit than other solutions. Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.
This implementation of the Marsaglia Engine is seeded by random.h default random engine. RandomEngine is only seeded once so its not a performance issue.
Example Use: Using Marsaglia Engine with RandomWalk

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

Example Use: Generating a random number with Marsaglia Engine

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

Definition at line 116 of file random.h.

7.9.2 Constructor & Destructor Documentation

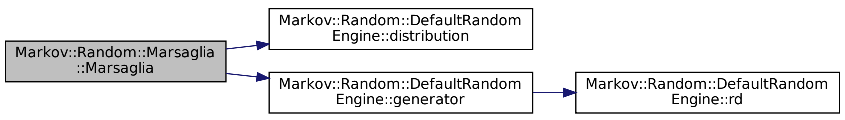
7.9.2.1 Marsaglia()

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

Construct Marsaglia Engine.
Initialize x,y and z using the default random engine.
Definition at line 123 of file random.h.

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

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


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

7.9.3 Member Function Documentation

7.9.3.1 distribution()

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

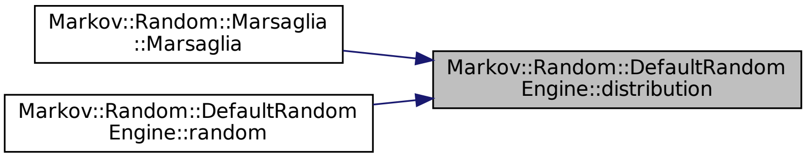
Distribution schema for seeding.

Definition at line 81 of file random.h.

```
                                                                          {
                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

7.9.3.2 generator()

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

Default random engine for seeding.

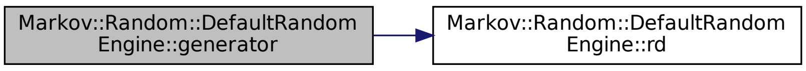
Definition at line 73 of file random.h.

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

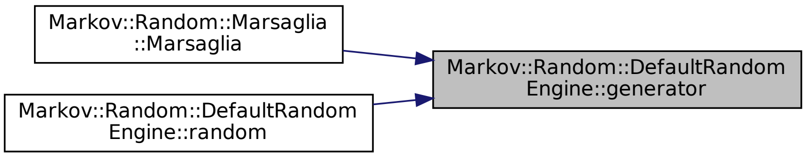
References Markov::Random::DefaultRandomEngine::rd().

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

7.9.3.3 random()

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

Generate Random Number.

Returns

random number in long range.

Reimplemented from Markov::Random::DefaultRandomEngine.

Definition at line 129 of file random.h.

```
                                        {
            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.

7.9.3.4 rd()

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

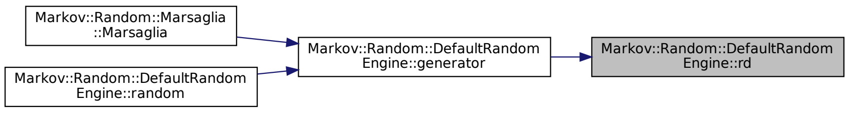
Default random device for seeding.

Definition at line 65 of file random.h.

```
                                        {
            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

7.9.4 Member Data Documentation

7.9.4.1 x

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

Definition at line 144 of file random.h.

Referenced by Marsaglia(), and random().

7.9.4.2 y

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

Definition at line 145 of file random.h.

Referenced by Marsaglia(), and random().

7.9.4.3 z

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

Definition at line 146 of file random.h.

Referenced by Marsaglia(), and random().

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

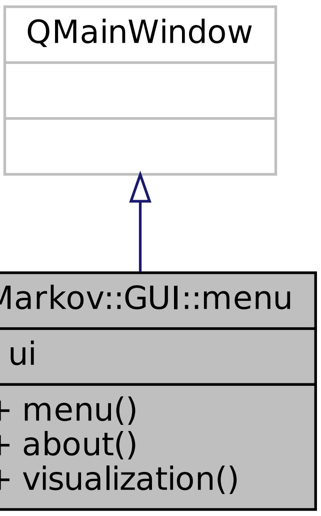
- random.h

7.10 Markov::GUI::menu Class Reference

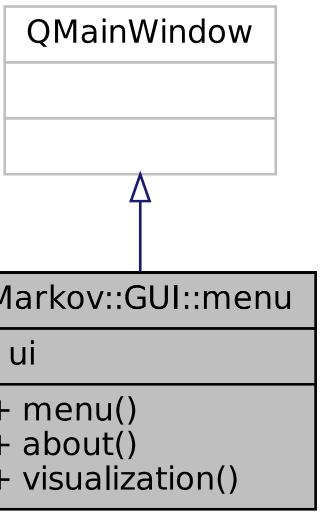
QT Menu class.

`#include <menu.h>`

Inheritance diagram for Markov::GUI::menu:



Collaboration diagram for Markov::GUI::menu:



Public Slots

- void about ()
- void visualization ()

Public Member Functions

- menu (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::main ui

7.10.1 Detailed Description

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

7.10.2 Constructor & Destructor Documentation

7.10.2.1 menu()

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

Definition at line 8 of file menu.cpp.

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


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

References ui.

7.10.3 Member Function Documentation

7.10.3.1 about

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

Definition at line 17 of file menu.cpp.

```
                  {


}
```

7.10.3.2 visualization

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

Definition at line 21 of file menu.cpp.

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

7.10.4 Member Data Documentation

7.10.4.1 ui

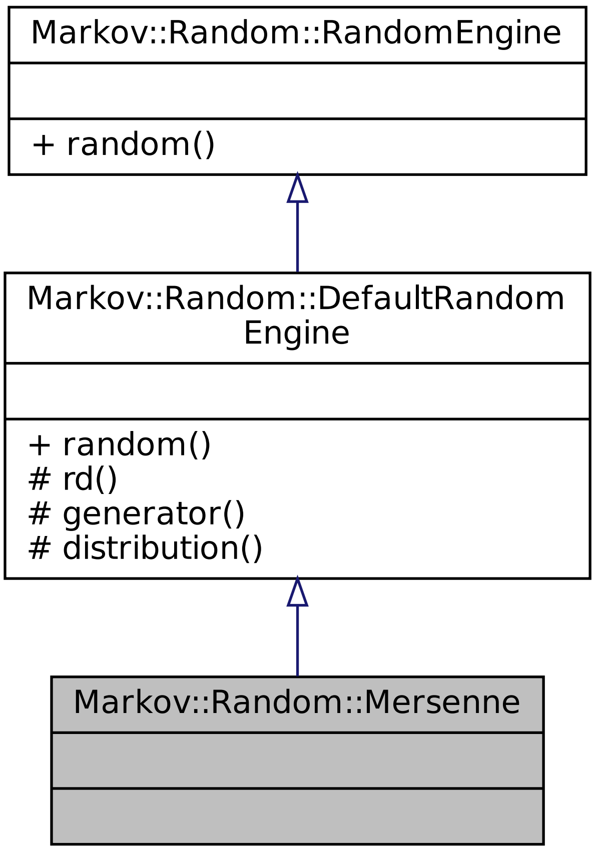
```
Ui::main Markov::GUI::menu::ui  [private]
```

Definition at line 15 of file menu.h.
Referenced by menu().
The documentation for this class was generated from the following files:

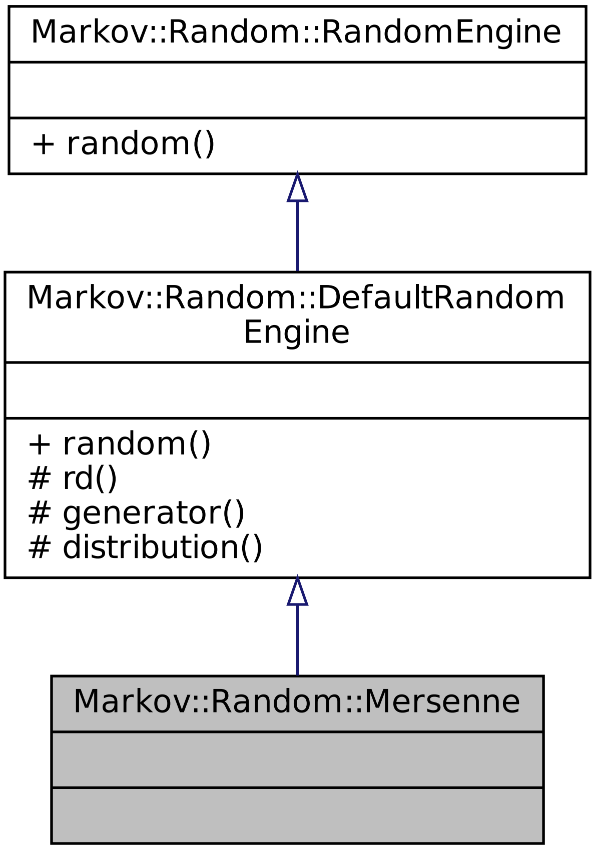
- menu.h
- menu.cpp

7.11 Markov::Random::Mersenne Class Reference

Implementation of Mersenne Twister Engine.
`#include <random.h>`
Inheritance diagram for Markov::Random::Mersenne:



Collaboration diagram for Markov::Random::Mersenne:



Public Member Functions

- unsigned long random ()

  Generate Random Number.

Protected Member Functions

- std::random_device & rd ()

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

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

  Distribution schema for seeding.

7.11.1 Detailed Description

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

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

Example Use: Generating a random number with Marsaglia Engine

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

Definition at line 174 of file random.h.

7.11.2 Member Function Documentation

7.11.2.1 distribution()

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

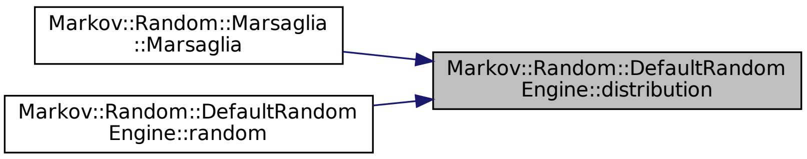
Distribution schema for seeding.

Definition at line 81 of file random.h.

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

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

7.11.2.2 generator()

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

Default random engine for seeding.

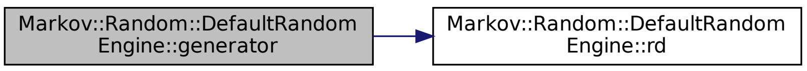
Definition at line 73 of file random.h.

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

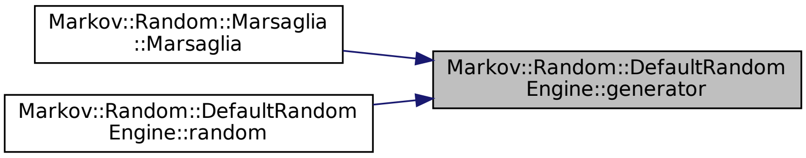
References Markov::Random::DefaultRandomEngine::rd().

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

7.11.2.3 random()

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

Generate Random Number.

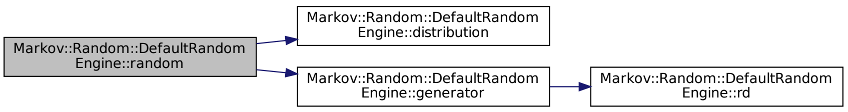
Returns

random number in long range.

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

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

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


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

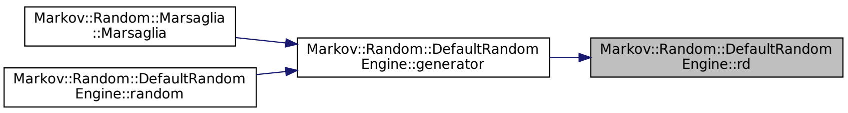
7.11.2.4 rd()

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

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

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

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


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:

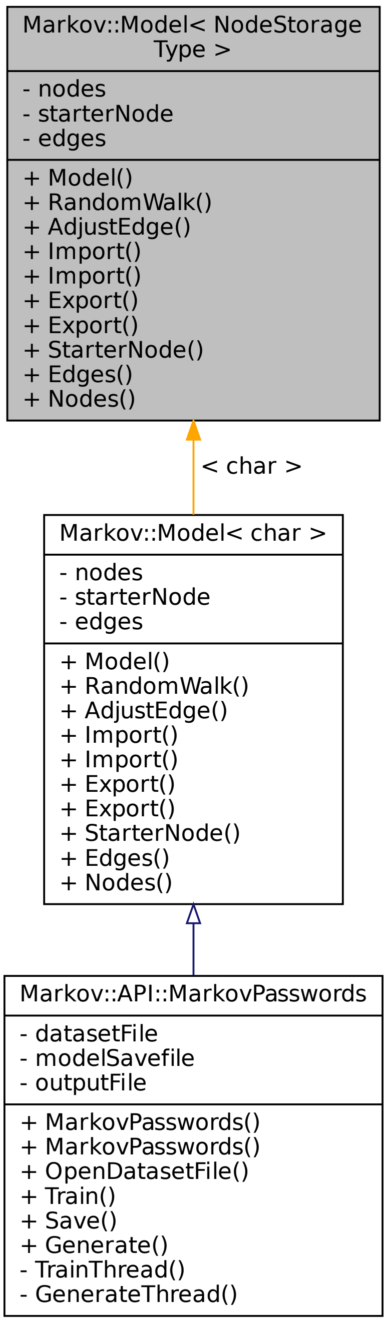
- random.h

7.12 Markov::Model< NodeStorageType > Class Template Reference

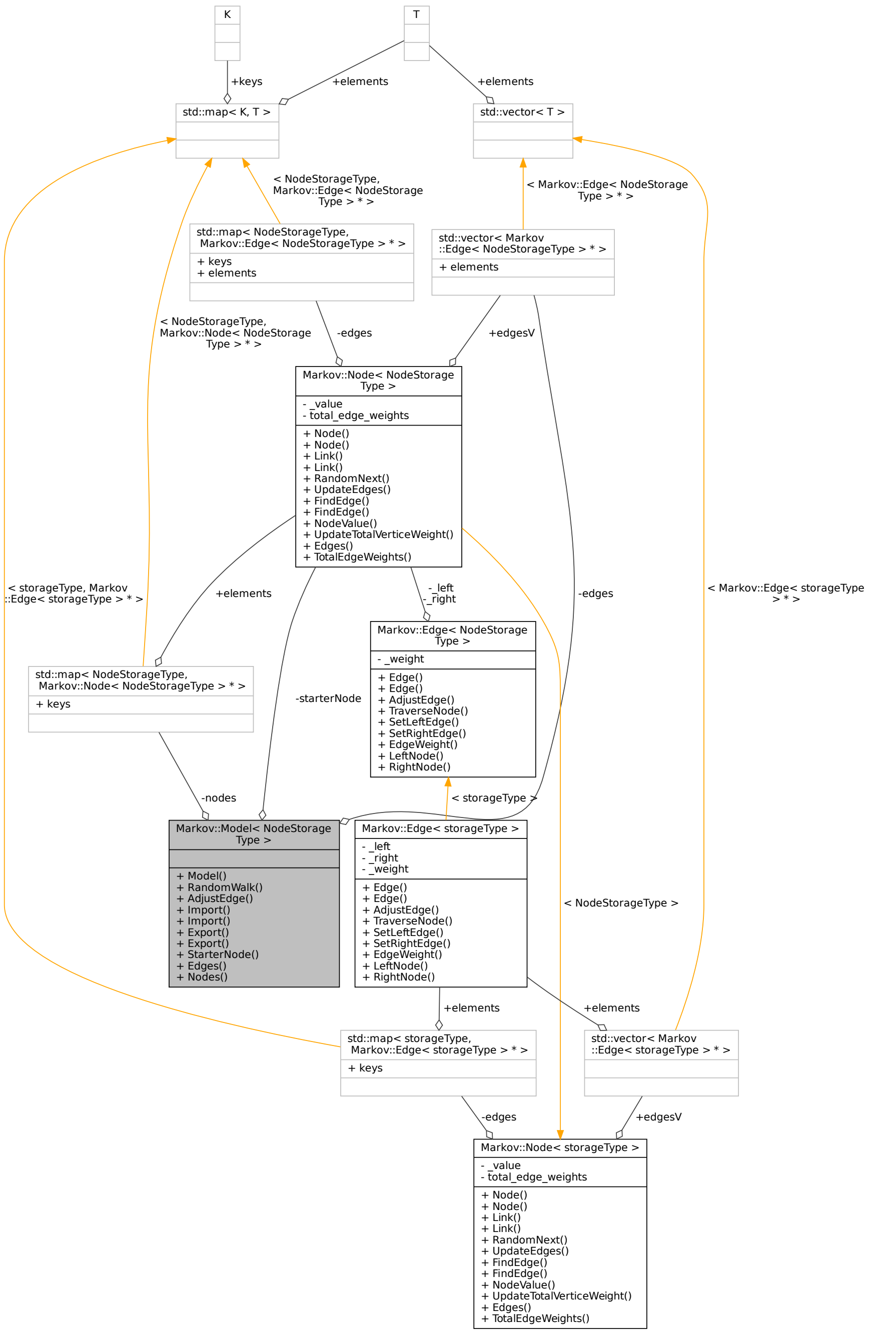
class for the final Markov Model, constructed from nodes and edges.

`#include <model.h>`

Inheritance diagram for Markov::Model< NodeStorageType >:



Collaboration diagram for Markov::Model< NodeStorageType >:



Public Member Functions

- [Model](#) ()

   Initialize a model with only start and end nodes.

- NodeStorageType ∗ RandomWalk (Markov::Random::RandomEngine ∗randomEngine, int minSetting, int maxSetting, NodeStorageType ∗buffer)

   Do a random walk on this model.

- void AdjustEdge (const NodeStorageType ∗payload, long int occurrence)

   Adjust the model with a single string.

- bool Import (std::ifstream ∗)

Import a file to construct the model.

- bool Import (const char ∗filename)

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

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

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

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

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

  Return starter Node.

Private Attributes

- std::map< NodeStorageType, Node< NodeStorageType > ∗ > nodes

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

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

  A list of all edges in this model.

7.12.1 Detailed Description

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

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

7.12.2 Constructor & Destructor Documentation

7.12.2.1 Model()

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

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

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

7.12.3 Member Function Documentation

7.12.3.1 AdjustEdge()

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

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

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

Parameters

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 }
```

7.12.3.2 Edges()

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

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 172 of file model.h.

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

7.12.3.3 Export() [1/2]

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

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

Returns

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

Example Use: Export file to filename

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

Definition at line 285 of file model.h.

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

7.12.3.4 Export() [2/2]

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

Export a file of the model.

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

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

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

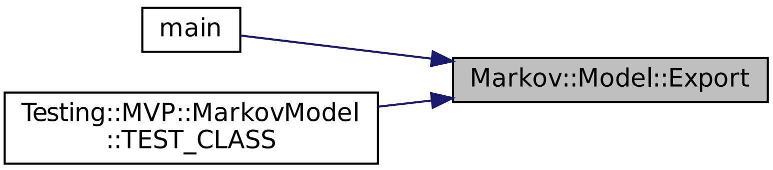
```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 273 of file model.h.

```
                                                         {
    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;
}
```

Referenced by main(), and Testing::MVP::MarkovModel::TEST_CLASS().

Here is the caller graph for this function:



7.12.3.5 Import() [1/2]

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

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

Returns

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

Example Use: Import a file with filename

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

Definition at line 265 of file model.h.

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

}
```

7.12.3.6 Import() [2/2]

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

Import a file to construct the model.

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

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

Returns

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

Example Use: Import a file from ifstream

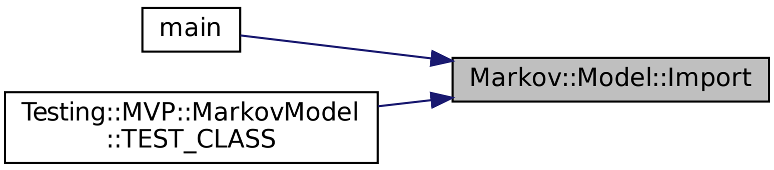
```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 206 of file model.h.

```
                                                            {
    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 main(), and Testing::MVP::MarkovModel::TEST_CLASS().

Here is the caller graph for this function:



7.12.3.7 Nodes()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 177 of file model.h.

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

7.12.3.8 RandomWalk()

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

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length 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

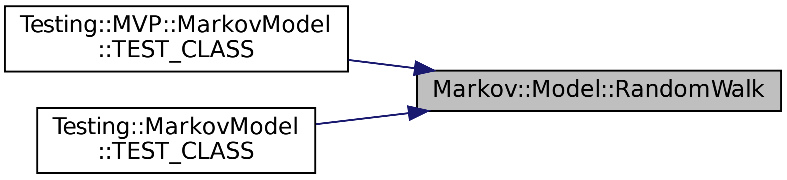
Null terminated string that was generated.

Definition at line 292 of file model.h.

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

Referenced by Testing::MVP::MarkovModel::TEST_CLASS(), and Testing::MarkovModel::TEST_CLASS().

Here is the caller graph for this function:


Testing::MVP::MarkovModel ::TEST_CLASS
Testing::MarkovModel ::TEST_CLASS
Markov::Model::RandomWalk

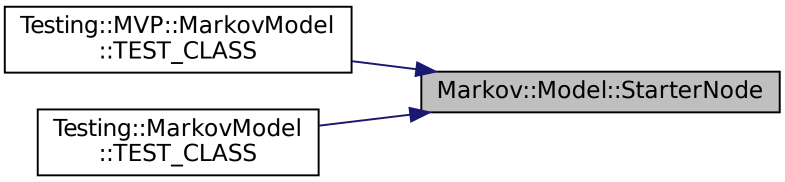
7.12.3.9 StarterNode()

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

Returns

starter node with 00 NodeValue

Definition at line 167 of file model.h.
```
00167 { return starterNode;}
```
Referenced by Testing::MVP::MarkovModel::TEST_CLASS(), and Testing::MarkovModel::TEST_CLASS().
Here is the caller graph for this function:


Testing::MVP::MarkovModel ::TEST_CLASS
Testing::MarkovModel ::TEST_CLASS
Markov::Model::StarterNode

7.12.4 Member Data Documentation

7.12.4.1 edges

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

7.12.4.2 nodes

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

7.12.4.3 starterNode

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

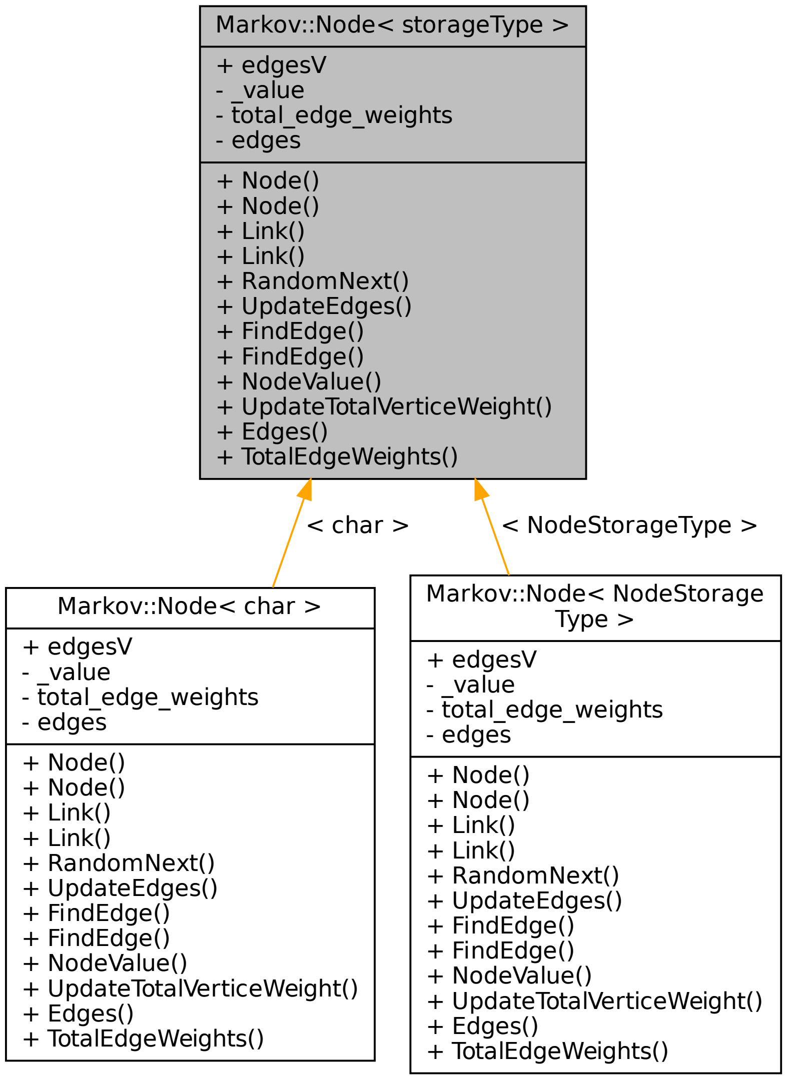
- model.h

7.13 Markov::Node< storageType > Class Template Reference

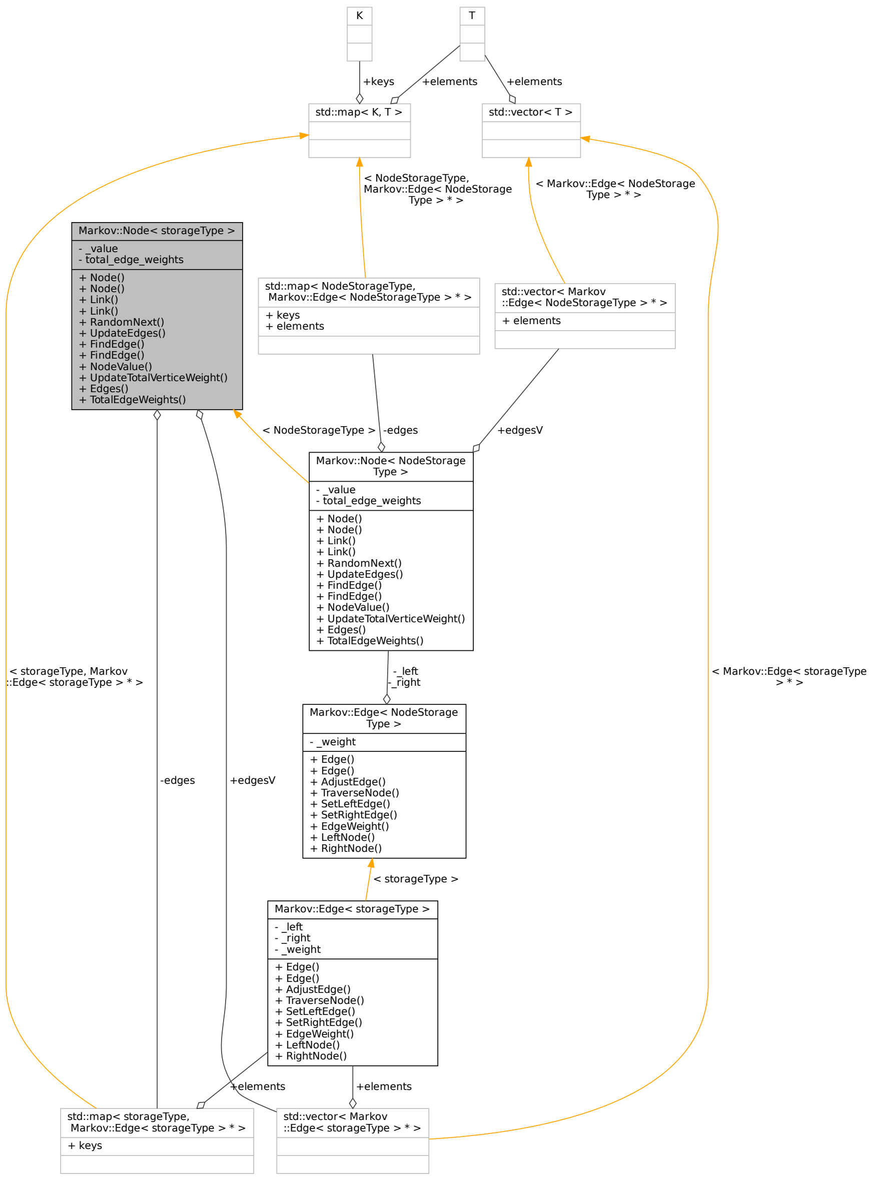
A node class that for the vertices of model. Connected with eachother using Edge.

`#include <edge.h>`

Inheritance diagram for Markov::Node< storageType >:



Collaboration diagram for Markov::Node< storageType >:



Public Member Functions

- Node ()

  Default constructor. Creates an empty Node.
- Node (storageType _value)

  Constructor. Creates a Node with no edges and with given NodeValue.
- Edge< storageType > ∗ Link (Node< storageType > ∗)

  Link this node with another, with this node as its source.
- Edge< storageType > ∗ Link (Edge< storageType > ∗)

  Link this node with another, with this node as its source.
- Node< storageType > ∗ RandomNext (Markov::Random::RandomEngine ∗randomEngine)

  Chose a random node from the list of edges, with regards to its EdgeWeight, and TraverseNode to that.
- bool UpdateEdges (Edge< storageType > ∗)

  Insert a new edge to the this.edges.

- Edge< storageType > ∗ FindEdge (storageType repr)

  Find an edge with its character representation.
- Edge< storageType > ∗ FindEdge (Node< storageType > ∗target)

  Find an edge with its pointer. Avoid unless 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;.

7.13.1 Detailed Description

template<typename storageType>
class Markov::Node< storageType >

A node class that for the vertices of model. Connected with eachother using Edge.
This class will later be templated to accept other data types than char∗.
Definition at line 8 of file edge.h.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 Node() [1/2]

```
template<typename storageType >
Markov::Node< storageType >::Node
```

Default constructor. Creates an empty Node.
Definition at line 196 of file node.h.

```
00196                              {
00197     this->_value = 0;
00198     this->total_edge_weights = 0L;
00199 };
```

7.13.2.2 Node() [2/2]

```
template<typename storageType >
Markov::Node< storageType >::Node (
                storageType _value )
```

Constructor. Creates a Node with no edges and with given NodeValue.

Parameters

_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;
};
```

7.13.3 Member Function Documentation

7.13.3.1 Edges()

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

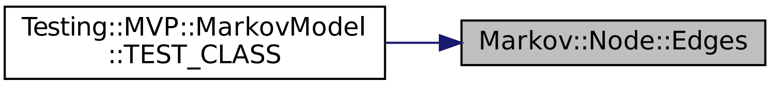
return edges

Definition at line 259 of file node.h.

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

Referenced by Testing::MVP::MarkovModel::TEST_CLASS().

Here is the caller graph for this function:



7.13.3.2 FindEdge() [1/2]

```
template<typename storageType >
Edge<storageType>* Markov::Node< storageType >::FindEdge (
            Node< storageType > * target )
```

Find an edge with its pointer. Avoid unless 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.

7.13.3.3 FindEdge() [2/2]

```
template<typename storageType >
```

```
Markov::Edge< storageType > * Markov::Node< storageType >::FindEdge (
              storageType repr )
```

Find an edge with its character representation.

Parameters

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

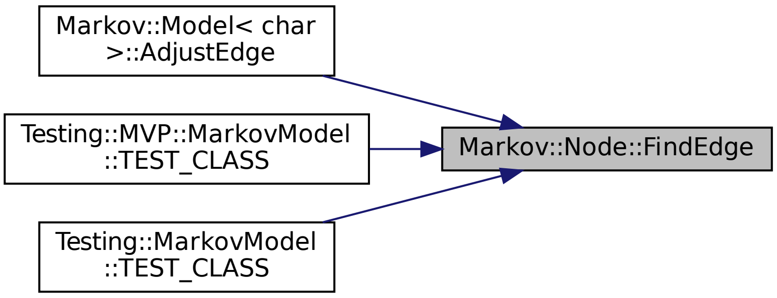
```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
Markov::Edge<unsigned char>* res = NULL;
src->Link(target1);
src->Link(target2);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

Definition at line 247 of file node.h.

```
                                                                              {
    auto e = this->edges.find(repr);
    if (e == this->edges.end()) return NULL;
    return e->second;
};
```

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

Here is the caller graph for this function:


Markov::Model< char >::AdjustEdge
Testing::MVP::MarkovModel ::TEST_CLASS
Testing::MarkovModel ::TEST_CLASS
Markov::Node::FindEdge

7.13.3.4 Link() [1/2]

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

Link this node with another, with this node as its source.
DOES NOT create a new Edge.

Parameters

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 }
```

7.13.3.5 Link() [2/2]

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

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

Creates a new Edge.

Parameters

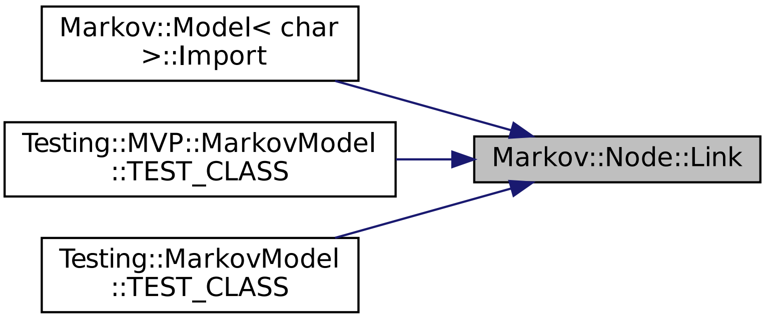
target	- Target node which will be the RightNode() of new edge.

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.
```
00207                                                                               {
00208     Markov::Edge<storageType>* v = new Markov::Edge<storageType>(this, n);
00209     this->UpdateEdges(v);
00210     return v;
00211 }
```
Referenced by Markov::Model< char >::Import(), Testing::MVP::MarkovModel::TEST_CLASS(), and Testing::MarkovModel::TEST_C

Here is the caller graph for this function:


Markov::Model< char >::Import
Testing::MVP::MarkovModel ::TEST_CLASS
Testing::MarkovModel ::TEST_CLASS
Markov::Node::Link

7.13.3.6 NodeValue()

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

Return character representation of this node.

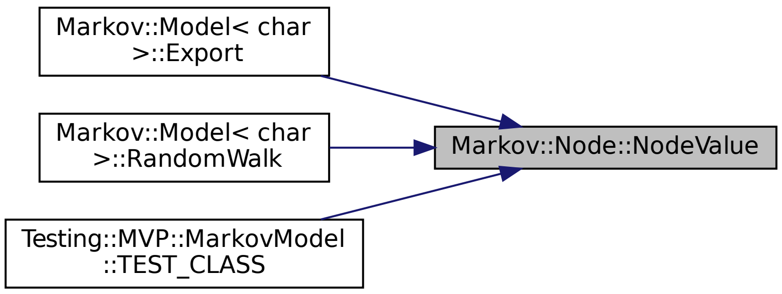
Returns

character representation at _value.

Definition at line 202 of file node.h.

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

Referenced by Markov::Model< char >::Export(), Markov::Model< char >::RandomWalk(), and Testing::MVP::MarkovModel::TEST_
Here is the caller graph for this function:


Markov::Model< char >::Export
Markov::Model< char >::RandomWalk
Testing::MVP::MarkovModel ::TEST_CLASS
Markov::Node::NodeValue

7.13.3.7 RandomNext()

```
template<typename storageType >
Markov::Node< storageType > * Markov::Node< storageType >::RandomNext (
              Markov::Random::RandomEngine * randomEngine )
```

Chose a random node from the list of edges, with regards to its EdgeWeight, and TraverseNode to that.
This operation is done by generating a random number in range of 0-this.total_edge_weights, and then iterating over the list of edges. At each step, EdgeWeight of the edge is subtracted from the random number, and once it is 0, next node is selected.

Returns

Node that was chosen at EdgeWeight biased random.

Example Use: Use randomNext to do a random walk on the model

```
char* buffer[64];
Markov::Model<char> model;
model.Import("model.mdl");
   Markov::Node<char>* n = model.starterNode;
int len = 0;
Markov::Node<char>* temp_node;
while (true) {
   temp_node = n->RandomNext(randomEngine);
   if (len >= maxSetting) {
       break;
   }
   else if ((temp_node == NULL) && (len < minSetting)) {
       continue;
   }
   else if (temp_node == NULL){
       break;
   }

   n = temp_node;
```

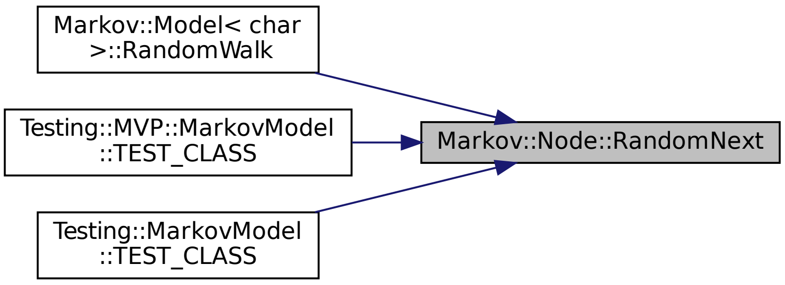
```
    buffer[len++] = n->NodeValue();
}
```

Definition at line 221 of file node.h.

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

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

Here is the caller graph for this function:


Markov::Model< char >::RandomWalk
Testing::MVP::MarkovModel ::TEST_CLASS
Testing::MarkovModel ::TEST_CLASS
Markov::Node::RandomNext

7.13.3.8 TotalEdgeWeights()

```
template<typename storageType >
long int Markov::Node< storageType >::TotalEdgeWeights  [inline]
```

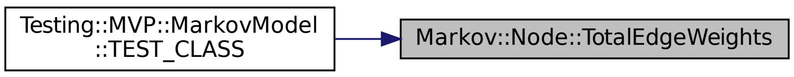
return total edge weights

Definition at line 264 of file node.h.

```
00264                                                                  {
00265     return this->total_edge_weights;
00266 }
```

Referenced by Testing::MVP::MarkovModel::TEST_CLASS().

Here is the caller graph for this function:


Testing::MVP::MarkovModel ::TEST_CLASS
Markov::Node::TotalEdgeWeights

7.13.3.9 UpdateEdges()

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

Insert a new edge to the this.edges.

Parameters

edge	- New edge that will be inserted.

Returns

true if insertion was successful, false if it fails.

Example Use: Construct and update edges

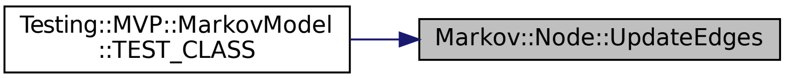
```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, 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();
}
```

Referenced by Testing::MVP::MarkovModel::TEST_CLASS().

Here is the caller graph for this function:


Testing::MVP::MarkovModel ::TEST_CLASS
Markov::Node::UpdateEdges

7.13.3.10 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;
}
```

7.13.4 Member Data Documentation

7.13.4.1 _value

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

Definition at line 169 of file node.h.

7.13.4.2 edges

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

Total weights of the vertices, required by RandomNext;.
A map of all edges connected to this node, where this node is at the LeftNode.
Map is indexed by unsigned char, which is the character representation of the node.
Definition at line 177 of file node.h.

7.13.4.3 edgesV

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

Definition at line 165 of file node.h.

7.13.4.4 total_edge_weights

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

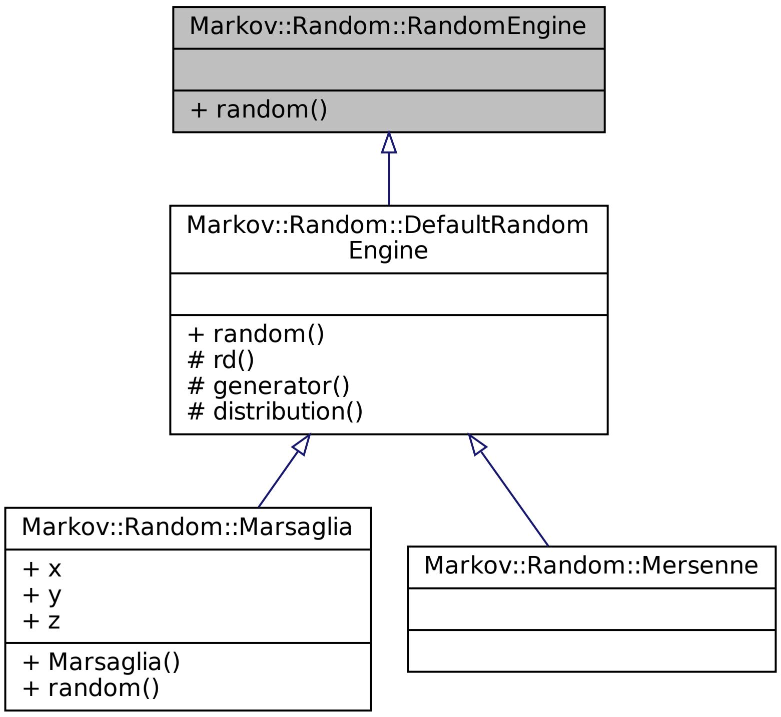
Character representation of this node. 0 for starter, 0xff for terminator.
Definition at line 171 of file node.h.
The documentation for this class was generated from the following files:

- edge.h
- node.h

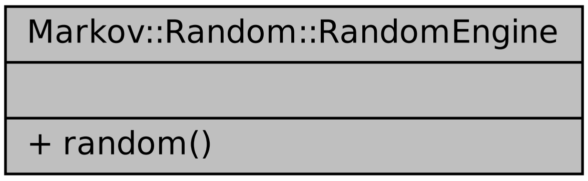
7.14 Markov::Random::RandomEngine Class Reference

An abstract class for Random Engine.
`#include <random.h>`

Inheritance diagram for Markov::Random::RandomEngine:



Collaboration diagram for Markov::Random::RandomEngine:



Public Member Functions

- virtual unsigned long random ()=0

7.14.1 Detailed Description

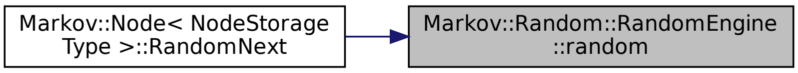
An abstract class for Random Engine.
This class is used for generating random numbers, which are used for random walking on the graph.
Main reason behind allowing different random engines is that some use cases may favor performance, while some favor good random.
Mersenne can be used for truer random, while Marsaglia can be used for deterministic but fast random.
Definition at line 21 of file random.h.

7.14.2 Member Function Documentation

7.14.2.1 random()

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

Implemented in Markov::Random::Marsaglia, and Markov::Random::DefaultRandomEngine.
Referenced by Markov::Node< NodeStorageType >::RandomNext().
Here is the caller graph for this function:


Markov::Node< NodeStorage Type >::RandomNext
Markov::Random::RandomEngine ::random

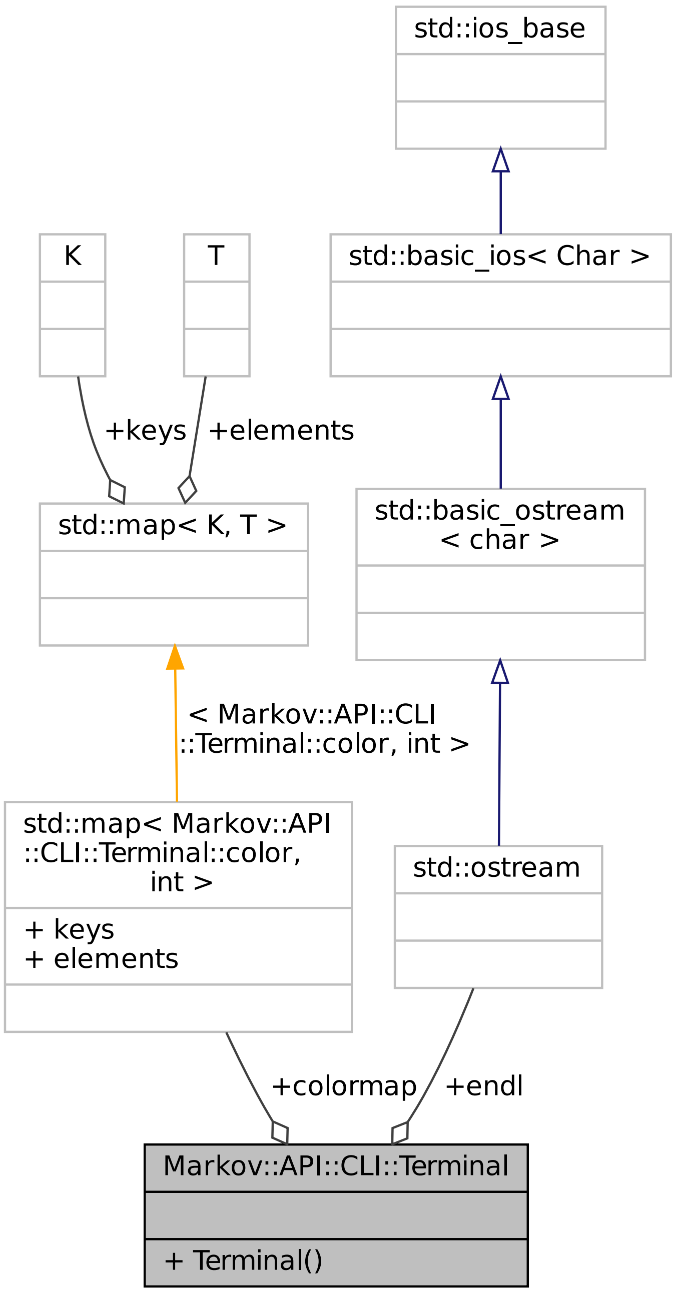
The documentation for this class was generated from the following file:

- random.h

7.15 Markov::API::CLI::Terminal Class Reference

pretty colors for Terminal. Windows Only.
`#include <term.h>`

Collaboration diagram for Markov::API::CLI::Terminal:


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

7.15.1 Detailed Description

pretty colors for Terminal. Windows Only.
Definition at line 18 of file term.h.

7.15.2 Member Enumeration Documentation

7.15.2.1 color

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

Enumerator

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

Definition at line 26 of file term.h.

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

7.15.3 Constructor & Destructor Documentation

7.15.3.1 Terminal()

`Terminal::Terminal ( )`

Default constructor. Get references to stdout and stderr handles.
Definition at line 56 of file term.cpp.

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

7.15.4 Member Data Documentation

7.15.4.1 colormap

```
std::map< Terminal::color, int > Terminal::colormap  [static]
```

Initial value:

```
= {
    {Terminal::color::BLACK, 30},
    {Terminal::color::BLUE, 34},
    {Terminal::color::GREEN, 32},
    {Terminal::color::CYAN, 36},
    {Terminal::color::RED, 31},
    {Terminal::color::MAGENTA, 35},
    {Terminal::color::BROWN, 0},
    {Terminal::color::LIGHTGRAY, 0},
    {Terminal::color::DARKGRAY, 0},
    {Terminal::color::YELLOW, 33},
    {Terminal::color::WHITE, 37},
    {Terminal::color::RESET, 0},
}
```

Definition at line 32 of file term.h.
Referenced by Markov::API::CLI::operator<<().

7.15.4.2 endl

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

Definition at line 37 of file term.h.
The documentation for this class was generated from the following files:

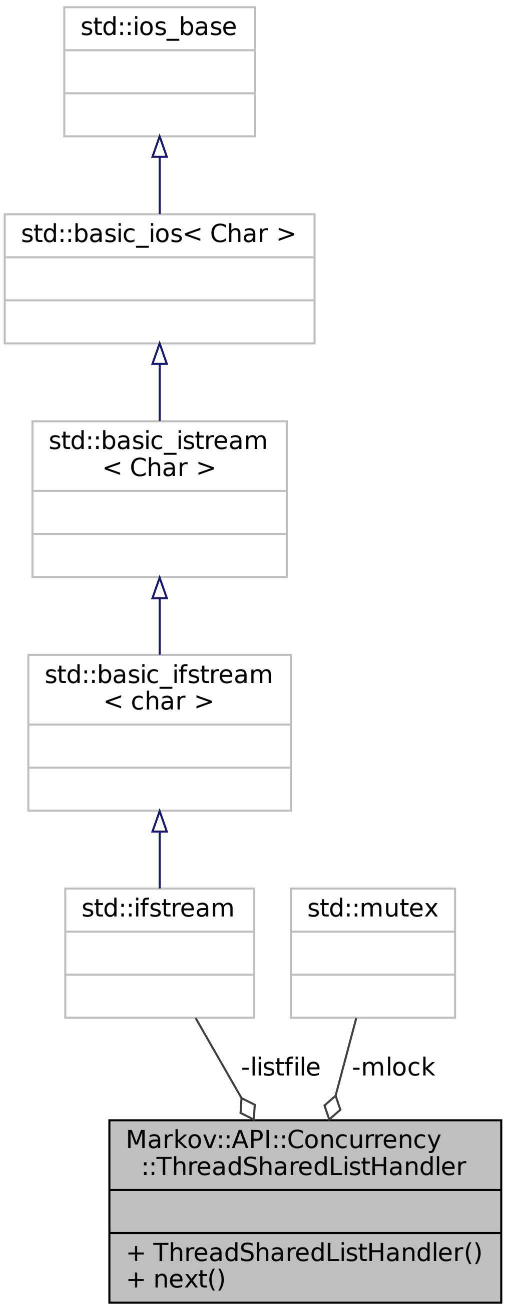
- term.h
- term.cpp

7.16 Markov::API::Concurrency::ThreadSharedListHandler Class Reference

Simple class for managing shared access to file.

```
#include <threadSharedListHandler.h>
```

Collaboration diagram for Markov::API::Concurrency::ThreadSharedListHandler:


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

7.16.1 Detailed Description

Simple class for managing shared access to file.
This class maintains the handover of each line from a file to multiple threads.
When two different threads try to read from the same file while reading a line isn't completed, it can have unexpected results. Line might be split, or might be read twice. This class locks the read action on the list until a line is completed, and then proceeds with the handover.
Definition at line 18 of file threadSharedListHandler.h.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 ThreadSharedListHandler()

```
Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler (
                const char * filename )
```

Construct the Thread Handler with a filename.
Simply open the file, and initialize the locks.
Example Use: Simple file read

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

Example Use: Example use case from MarkovPasswords showing multithreaded access

```
void MarkovPasswords::Train(const char* datasetFileName, char delimiter, int threads)   {
     ThreadSharedListHandler listhandler(datasetFileName);
     auto start = std::chrono::high_resolution_clock::now();
     std::vector<std::thread*> threadsV;
     for(int i=0;i<threads;i++){
         threadsV.push_back(new std::thread(&MarkovPasswords::TrainThread, this, &listhandler,
       datasetFileName, delimiter));
     }
    for(int i=0;i<threads;i++){
         threadsV[i]->join();
         delete threadsV[i];
     }
     auto finish = std::chrono::high_resolution_clock::now();
     std::chrono::duration<double> elapsed = finish - start;
     std::cout « "Elapsed time: " « elapsed.count() « " s\n";
 }
 void MarkovPasswords::TrainThread(ThreadSharedListHandler *listhandler, const char* datasetFileName, char
       delimiter){
     char format_str[] ="%ld,%s";
     format_str[2]=delimiter;
     std::string line;
     while (listhandler->next(&line)) {
         long int oc;
         if (line.size() > 100) {
             line = line.substr(0, 100);
         }
         char* linebuf = new char[line.length()+5];
         sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
         this->AdjustEdge((const char*)linebuf, oc);
         delete linebuf;
     }
 }
```

Parameters

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.

7.16.3 Member Function Documentation

7.16.3.1 next()

```
bool Markov::API::Concurrency::ThreadSharedListHandler::next (
                std::string * line )
```

Read the next line from the file.

This action will be blocked until another thread (if any) completes the read operation on the file.

Example Use: Simple file read

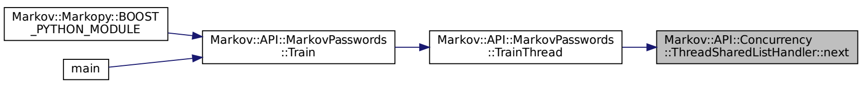
```
ThreadSharedListHandler listhandler("test.txt");
std::string line;
std::cout « listhandler->next(&line) « "\n";
```

Definition at line 10 of file threadSharedListHandler.cpp.

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

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

Here is the caller graph for this function:


Markov::Markopy::BOOST_PYTHON_MODULE
main
Markov::API::MarkovPasswords::Train
Markov::API::MarkovPasswords::TrainThread
Markov::API::Concurrency::ThreadSharedListHandler::next

7.16.4 Member Data Documentation

7.16.4.1 listfile

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

Definition at line 88 of file threadSharedListHandler.h.

Referenced by ThreadSharedListHandler().

7.16.4.2 mlock

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

Definition at line 89 of file threadSharedListHandler.h.

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

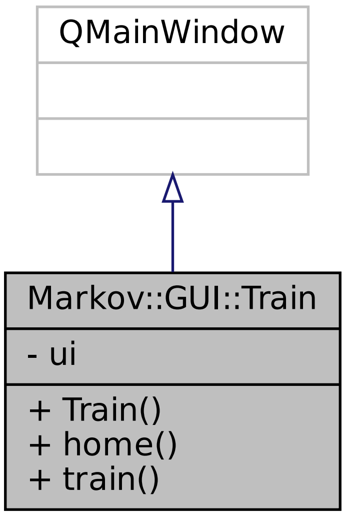
- threadSharedListHandler.h
- threadSharedListHandler.cpp

7.17 Markov::GUI::Train Class Reference

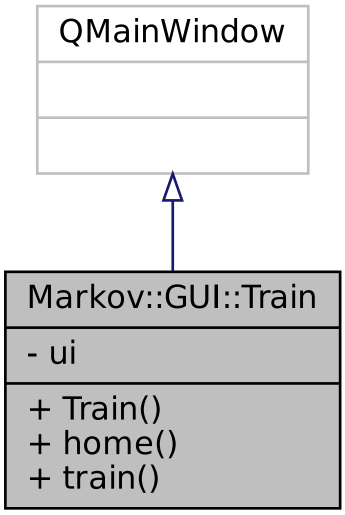
QT Training page class.

```
#include <Train.h>
```

Inheritance diagram for Markov::GUI::Train:



Collaboration diagram for Markov::GUI::Train:



Public Slots

- void home ()
- void train ()

Public Member Functions

- Train (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::Train ui

7.17.1 Detailed Description

QT Training page class.
Definition at line 9 of file Train.h.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 Train()

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

7.17.3 Member Function Documentation

7.17.3.1 home

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

7.17.3.2 train

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

7.17.4 Member Data Documentation

7.17.4.1 ui

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

Definition at line 15 of file Train.h.
The documentation for this class was generated from the following file:

- Train.h

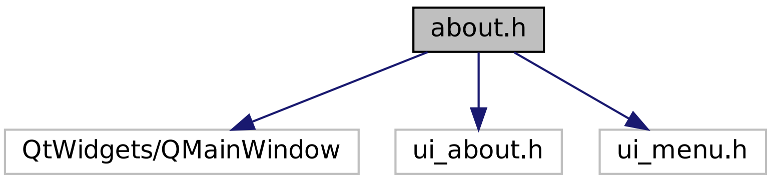
Chapter 8

File Documentation

8.1 about.h File Reference

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

Include dependency graph for about.h:



Classes

- class Markov::GUI::about

  QT Class for about page.

Namespaces

- Markov

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

  namespace for MarkovPasswords API GUI wrapper

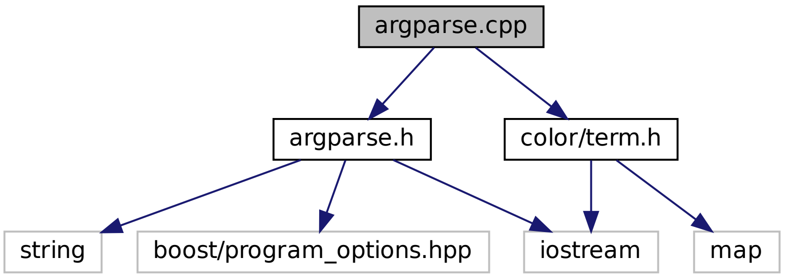
8.2 about.h

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

namespace Markov::GUI{

    class about :public QMainWindow {
    Q_OBJECT
    public:
        about(QWidget* parent = Q_NULLPTR);

    private:
        Ui:: main ui;


    };
};


```

8.3 argparse.cpp File Reference

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

Include dependency graph for argparse.cpp:



8.4 argparse.cpp

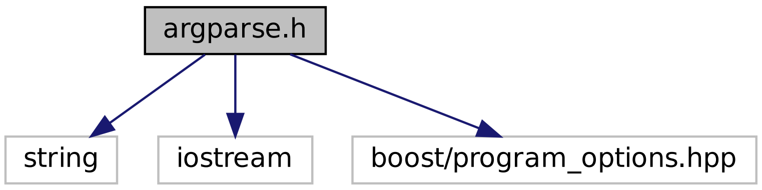
```
#include "argparse.h"
#include "color/term.h"

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;
}

```

8.5 argparse.h File Reference

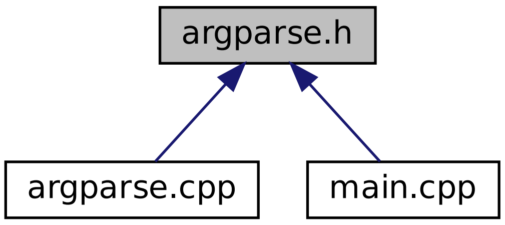
```
#include <string>
```

```
#include <iostream>
#include <boost/program_options.hpp>
```

Include dependency graph for argparse.h:



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



Classes

- struct Markov::API::CLI::_programOptions

  Structure to hold parsed cli 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.

8.5.1 Macro Definition Documentation

8.5.1.1 BOOST_ALL_DYN_LINK

```
#define BOOST_ALL_DYN_LINK 1
```

Definition at line 4 of file argparse.h.

8.6 argparse.h

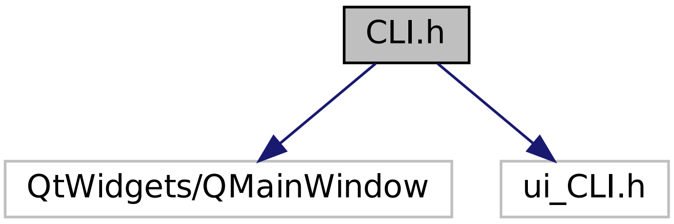
```
00001 #include<string>
00002 #include<iostream>
00003 
00004 #define BOOST_ALL_DYN_LINK 1
00005 
00006 #include <boost/program_options.hpp>
00009 namespace opt = boost::program_options;
00010 
00014 namespace Markov::API::CLI{
00015 
00018     typedef struct _programOptions {
00019         bool bImport;
00020         bool bExport;
00021         bool bFailure;
00022         char seperator;
00023         std::string importname;
00024         std::string exportname;
00025         std::string wordlistname;
00026         std::string outputfilename;
00027         std::string datasetname;
00028         int generateN;
00029     } ProgramOptions;
00030 
00031 
00034     class Argparse {
00035     public:
00036 
00037         Argparse();
00038 
00046         Argparse(int argc, char** argv) {
00047 
00048             /*bool bImp;
00049             bool bExp;
00050             bool bFail;
00051             char sprt;
00052             std::string imports;
00053             std::string exports;
00054             std::string outputs;
00055             std::string datasets;
00056             int generateN;
00057             */
00058             opt::options_description desc("Options");
00059 
00060 
00061             desc.add_options()
00062                 ("generate", "Generate strings with given parameters")
00063                 ("train", "Train model with given parameters")
00064                 ("combine", "Combine")
00065                 ("import", opt::value<std::string>(), "Import model file")
00066                 ("output", opt::value<std::string>(), "Output model file. This model will be exported when done. Will be ignored for generation mode")
00067                 ("dataset", opt::value<std::string>(), "Dataset file to read input from training. Will be ignored for generation mode")
00068                 ("seperator", opt::value<char>(), "Seperator character to use with training data. (character between occurence and value)")
00069                 ("wordlist", opt::value<std::string>(), "Wordlist file path to export generation results to. Will be ignored for training mode")
00070                 ("count", opt::value<int>(), "Number of lines to generate. Ignored in training mode")
00071                 ("verbosity", "Output verbosity")
00072                 ("help", "Option definitions");
00073 
00074                 opt::variables_map vm;
00075 
```

```
                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;
        }

        Markov::API::CLI::ProgramOptions getProgramOptions(void) {
            return this->po;
        }

        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);*/

        }

        static Markov::API::CLI::ProgramOptions* parse(int argc, char** argv);


        static void help();

    private:
        Markov::API::CLI::ProgramOptions po;
    };

};

```

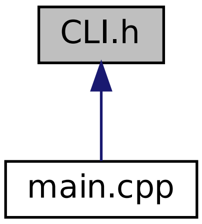
8.7 CLI.h File Reference

```
#include <QtWidgets/QMainWindow>
#include "ui_CLI.h"
```

Include dependency graph for CLI.h:



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



Classes

- class Markov::GUI::CLI

    QT CLI Class.

Namespaces

- Markov

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

- Markov::GUI

    namespace for MarkovPasswords API GUI wrapper

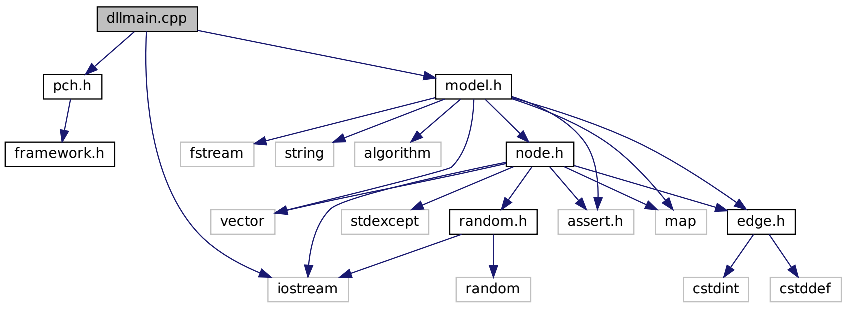
8.8 CLI.h

```
#pragma once
#include <QtWidgets/QMainWindow>
#include "ui_CLI.h"

namespace Markov::GUI{
    class CLI :public QMainWindow {
        Q_OBJECT
    public:
        CLI(QWidget* parent = Q_NULLPTR);

    private:
        Ui::CLI ui;

    public slots:
        void start();
        void statistics();
        void about();
    };
};
```

8.9 dllmain.cpp File Reference

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

Include dependency graph for dllmain.cpp:



8.10 dllmain.cpp

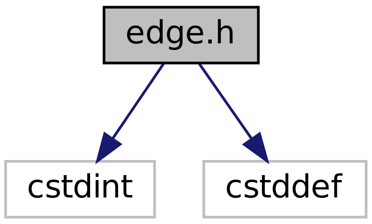
```
#include "pch.h"
#include "model.h"
#include <iostream>


#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

```

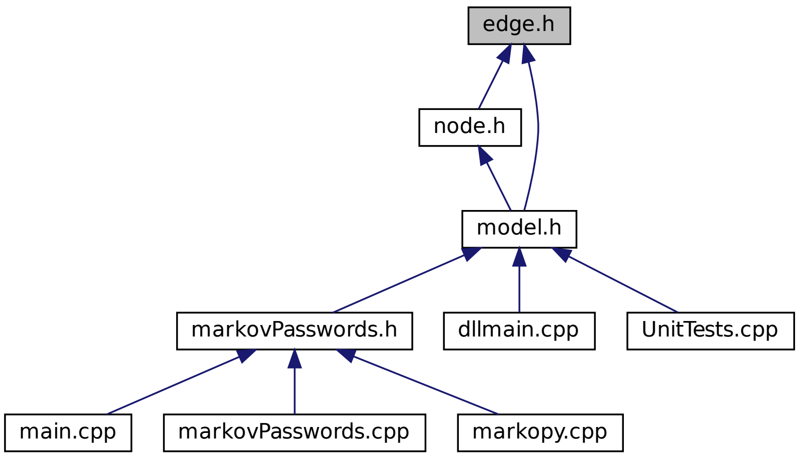
8.11 edge.h File Reference

`#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.

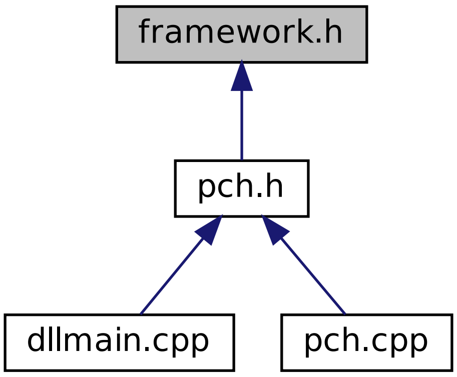
8.12 edge.h

```
#pragma once
#include <cstdint>
#include <cstddef>

namespace Markov {

    template <typename NodeStorageType>
    class Node;

    template <typename NodeStorageType>
    class Edge {
    public:

        Edge<NodeStorageType>();

        Edge<NodeStorageType>(Node<NodeStorageType>* _left, Node<NodeStorageType>* _right);

        void AdjustEdge(long int offset);

        inline Node<NodeStorageType>* TraverseNode();

        void SetLeftEdge (Node<NodeStorageType>*);
        void SetRightEdge(Node<NodeStorageType>*);

        inline uint64_t EdgeWeight();

        Node<NodeStorageType>* LeftNode();

        inline Node<NodeStorageType>* RightNode();

    private:
        Node<NodeStorageType>* _left;
        Node<NodeStorageType>* _right;
        long int _weight;
    };


};

//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;
}


```

8.13 framework.h File Reference

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



Macros

- #define WIN32_LEAN_AND_MEAN

8.13.1 Macro Definition Documentation

8.13.1.1 WIN32_LEAN_AND_MEAN

```
#define WIN32_LEAN_AND_MEAN
```

Definition at line 3 of file framework.h.

8.14 framework.h

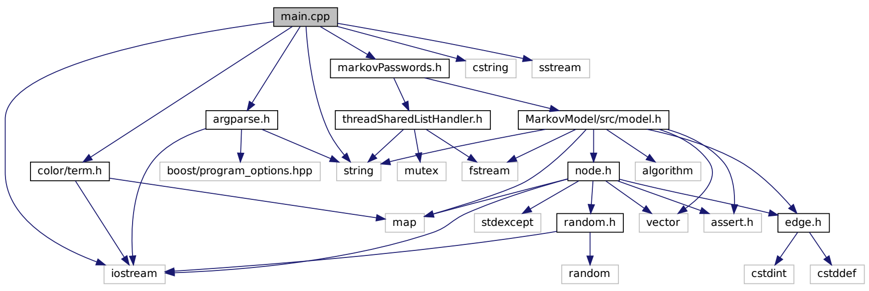
```
00001 #pragma once
00002
00003 #define WIN32_LEAN_AND_MEAN             // Exclude rarely-used stuff from Windows headers
00004 // Windows Header Files
```

```
00005
00006 #ifdef _WIN32
00007 #include <windows.h>
00008 #endif
00009
```

8.15 main.cpp File Reference

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

Include dependency graph for src/main.cpp:



Functions

- int main (int argc, char **argv)

  Launch CLI tool.

8.15.1 Function Documentation

8.15.1.1 main()

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

Launch CLI tool.

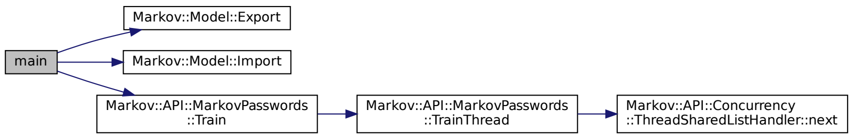
Definition at line 13 of file src/main.cpp.

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

```
    std::cout « "Generation done....\n";
    return 0;
}
```

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

Here is the call graph for this function:


main
Markov::Model::Export
Markov::Model::Import
Markov::API::MarkovPasswords ::Train
Markov::API::MarkovPasswords ::TrainThread
Markov::API::Concurrency ::ThreadSharedListHandler::next

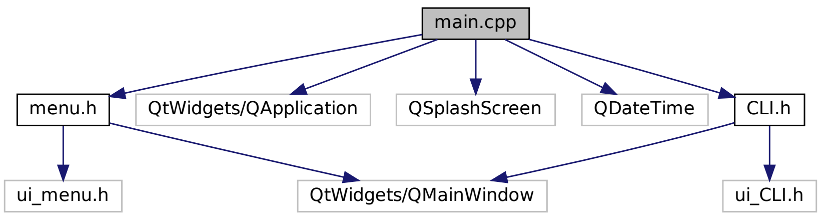
8.16 src/main.cpp

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


int main(int argc, char** argv) {

    Markov::API::CLI::Terminal t;
    /*
    ProgramOptions* p  = Argparse::parse(argc, argv);

    if (p==0 || p->bFailure) {
        std::cout « TERM_FAIL « "Arguments Failed to Parse" « std::endl;
        Argparse::help();
    }*/
    Markov::API::CLI::Argparse a(argc,argv);

    Markov::API::MarkovPasswords markovPass;
    std::cout « "Importing model.\n";
    markovPass.Import("models/2gram.mdl");
    std::cout « "Import done. Training...\n";
    markovPass.Train("datasets/graduation.corpus", '\t', 50);
    std::cout « "Training done. Exporting to file.\n";
    markovPass.Export("models/finished2.mdl");

    std::cout « "Generation done....\n";
    return 0;
}

```

8.17 main.cpp File Reference

```
#include "menu.h"
#include <QtWidgets/QApplication>
#include <QSplashScreen>
#include <QDateTime>
#include "CLI.h"
```

Include dependency graph for UI/src/main.cpp:



Functions

- int main (int argc, char ∗argv[ ])

  Launch UI.

8.17.1 Function Documentation

8.17.1.1 main()

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

Launch UI.

Definition at line 12 of file UI/src/main.cpp.

```
{



    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();
}
```

8.18 UI/src/main.cpp

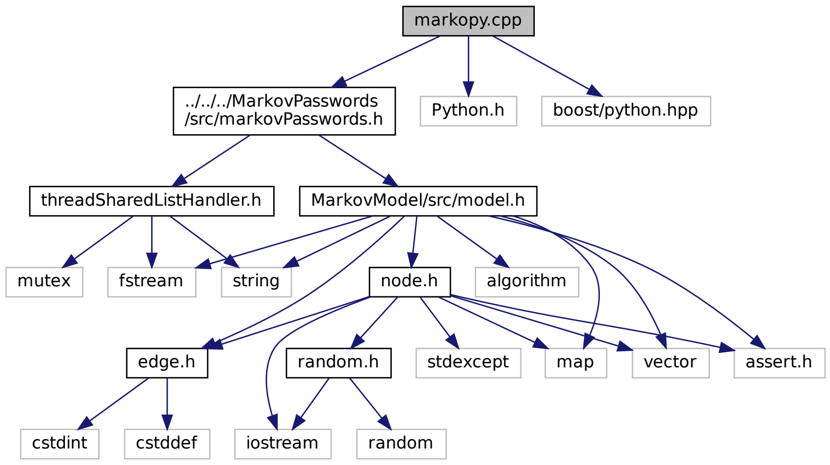
```
00001 //#include "MarkovPasswordsGUI.h"
00002 #include "menu.h"
00003 #include <QtWidgets/QApplication>
00004 #include <QSplashScreen>
00005 #include < QDateTime >
00006 #include "CLI.h"
00007
00008 using namespace Markov::GUI;
00009
00012 int main(int argc, char *argv[])
00013 {
00014
00015
00016
```

```
    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();
}

```

8.19 markopy.cpp File Reference

```
#include "../../../MarkovPasswords/src/markovPasswords.h"
#include <Python.h>
#include <boost/python.hpp>
```

Include dependency graph for markopy.cpp:



Namespaces

- Markov

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

Macros

- #define BOOST_PYTHON_STATIC_LIB

Functions

- Markov::Markopy::BOOST_PYTHON_MODULE (markopy)

8.19.1 Macro Definition Documentation

8.19.1.1 BOOST_PYTHON_STATIC_LIB

```
#define BOOST_PYTHON_STATIC_LIB
```

Definition at line 4 of file markopy.cpp.

8.20 markopy.cpp

```
00001 #pragma once
00002 #include "../../../MarkovPasswords/src/markovPasswords.h"
00003
00004 #define BOOST_PYTHON_STATIC_LIB
00005 #include <Python.h>
00006 #include <boost/python.hpp>
00007
00008 using namespace boost::python;
00009
00010 namespace Markov::Markopy{
00011     BOOST_PYTHON_MODULE(markopy)
00012     {
00013         bool (Markov::API::MarkovPasswords::*Import)(const char*) = &Markov::Model<char>::Import;
00014         bool (Markov::API::MarkovPasswords::*Export)(const char*) = &Markov::Model<char>::Export;
00015         class_<Markov::API::MarkovPasswords>("MarkovPasswords", init<>())
00016             .def(init<>())
00017             .def("Train", &Markov::API::MarkovPasswords::Train,
00018             "Train the model\n"
00019             "\n"
00020             ":param datasetFileName: Ifstream* to the dataset. If null, use class member\n"
00021             ":param delimiter: a character, same as the delimiter in dataset content\n"
00022             ":param threads: number of OS threads to spawn\n")
00023             .def("Generate", &Markov::API::MarkovPasswords::Generate,
00024             "Generate passwords from a trained model.\n"
00025             ":param n: Ifstream* to the dataset. If null, use class member\n"
00026             ":param wordlistFileName: a character, same as the delimiter in dataset content\n"
00027             ":param minLen: number of OS threads to spawn\n"
00028             ":param maxLen: Ifstream* to the dataset. If null, use class member\n"
00029             ":param threads: a character, same as the delimiter in dataset content\n"
00030             ":param threads: number of OS threads to spawn\n")
00031             .def("Import", Import, "Import a model file.")
00032             .def("Export", Export, "Export a model to file.")
00033         ;
00034     };
00035 };
00036
00037
```

8.21 markopy_cli.py File Reference

Namespaces

- markopy_cli

Functions

- def markopy_cli.cli_init (input_model)
- def markopy_cli.cli_train (model, dataset, 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

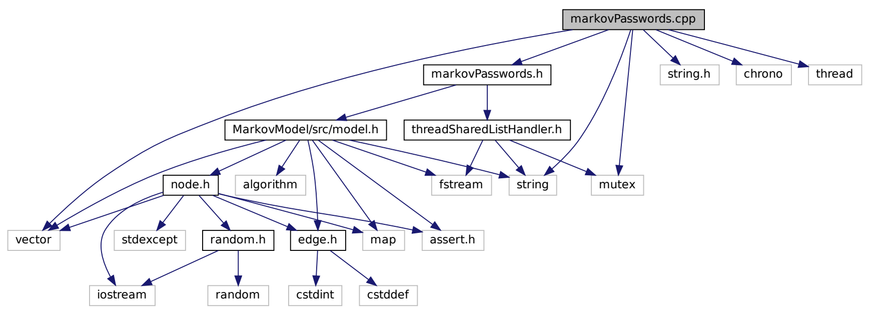
8.22 markopy_cli.py

```python
#!/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)
```

8.23 markovPasswords.cpp File Reference

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

Include dependency graph for markovPasswords.cpp:



8.24 markovPasswords.cpp

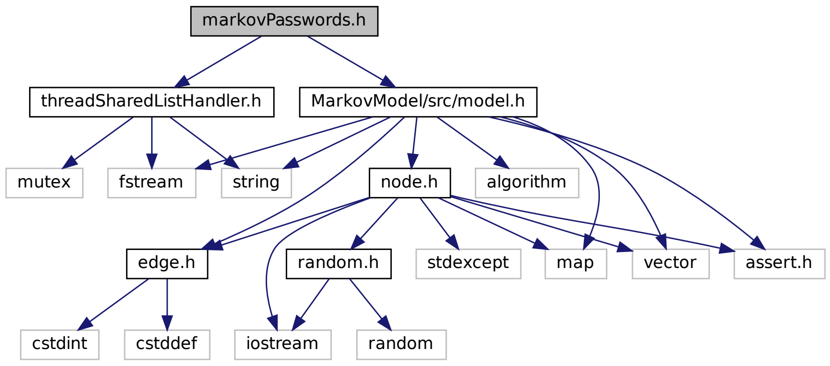
```
#pragma once
#include "markovPasswords.h"
#include <string.h>
```

```
#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();
    }
}
```

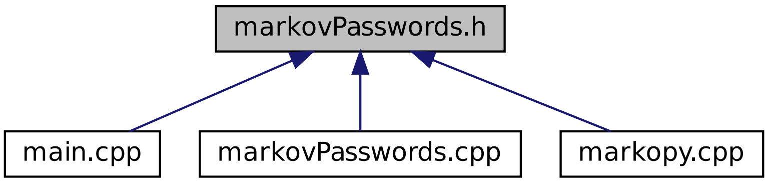
8.25 markovPasswords.h File Reference

```
#include "threadSharedListHandler.h"
#include "MarkovModel/src/model.h"
```

Include dependency graph for markovPasswords.h:



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



Classes

- class Markov::API::MarkovPasswords

  Markov::Model with char represented nodes.

Namespaces

- Markov

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

  Namespace for the MarkovPasswords API.

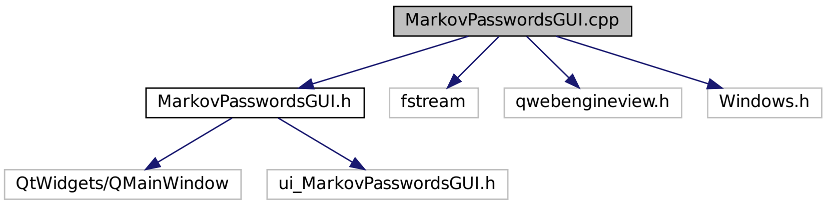
8.26 markovPasswords.h

```
#pragma once
#include "threadSharedListHandler.h"
#include "MarkovModel/src/model.h"


namespace Markov::API{

    class MarkovPasswords : public Markov::Model<char>{
    public:

        MarkovPasswords();

        MarkovPasswords(const char* filename);

        std::ifstream* OpenDatasetFile(const char* filename);


        void Train(const char* datasetFileName, char delimiter, int threads);



        std::ofstream* Save(const char* filename);

        void Generate(unsigned long int n, const char* wordlistFileName, int minLen=6, int maxLen=12,
 int threads=20);


    private:

        void TrainThread(Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char
delimiter);

        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;
    };



};

```

8.27 MarkovPasswordsGUI.cpp File Reference

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

Include dependency graph for MarkovPasswordsGUI.cpp:



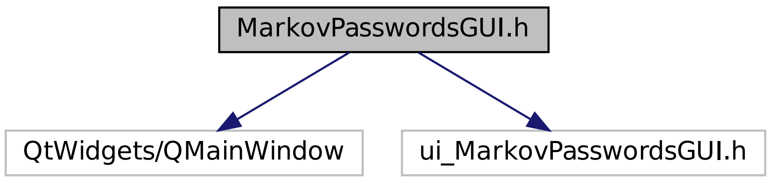
8.28 MarkovPasswordsGUI.cpp

```
#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

}
```

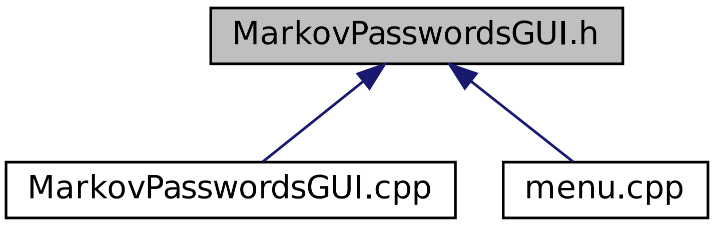
8.29 MarkovPasswordsGUI.h File Reference

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

Include dependency graph for MarkovPasswordsGUI.h:



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



Classes

- class Markov::GUI::MarkovPasswordsGUI

Reporting UI.

Namespaces

- Markov

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

  namespace for MarkovPasswords API GUI wrapper

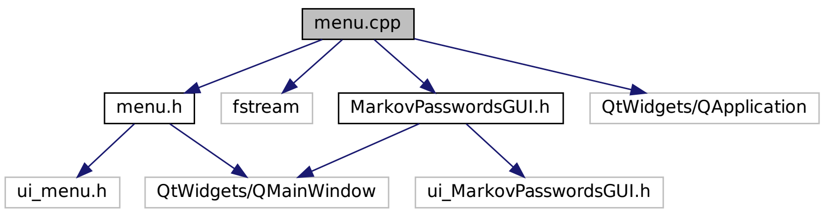
8.30 MarkovPasswordsGUI.h

```
#pragma once

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


namespace Markov::GUI{
    class MarkovPasswordsGUI : public QMainWindow {
        Q_OBJECT

    public:
        MarkovPasswordsGUI(QWidget *parent = Q_NULLPTR);

        void renderHTMLFile(std::string* filename);

        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();
    };
};


```

8.31 menu.cpp File Reference

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

Include dependency graph for menu.cpp:



8.32 menu.cpp

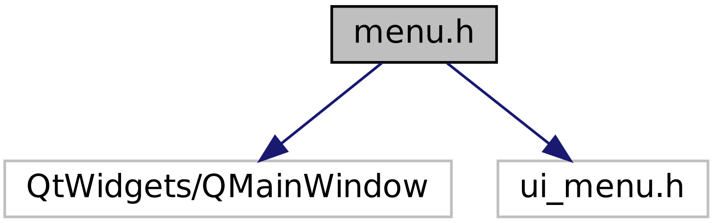
```
#include "menu.h"
```

```
#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();
}
```

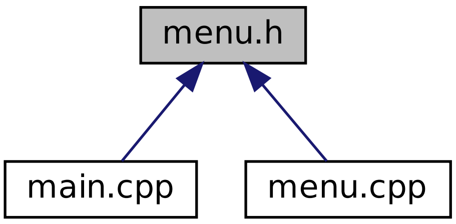
8.33 menu.h File Reference

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

Include dependency graph for menu.h:



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



Classes

- class Markov::GUI::menu

  QT Menu class.

Namespaces

- Markov

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

- Markov::GUI

  namespace for MarkovPasswords API GUI wrapper

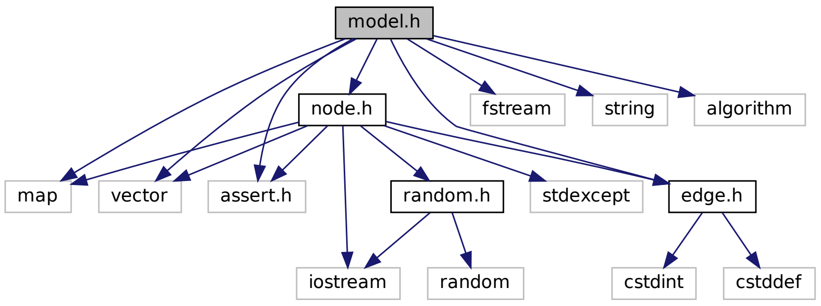
8.34 menu.h

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


namespace Markov::GUI{
    class menu:public QMainWindow {
    Q_OBJECT
    public:
        menu(QWidget* parent = Q_NULLPTR);

    private:
        Ui::main ui;

    public slots:
        void about();
        void visualization();
    };
};
```

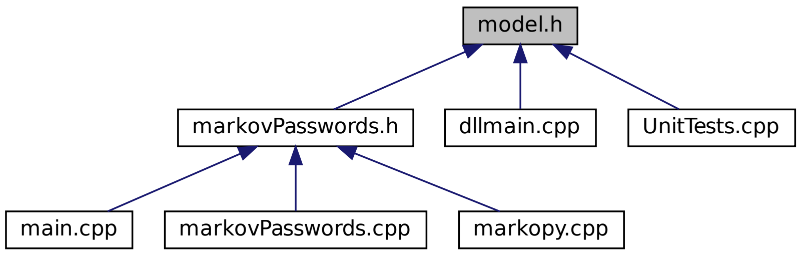
8.35 model.h File Reference

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

Include dependency graph for model.h:



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



Classes

- class Markov::Node< storageType >

  A node class that for the vertices of model. Connected with eachother using Edge.
- class Markov::Edge< NodeStorageType >

  Edge class used to link nodes in the model together.
- class Markov::Model< NodeStorageType >

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

Namespaces

- Markov

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

8.36 model.h

```
00001
00006 #pragma once
00007 #include <map>
00008 #include <vector>
00009 #include <fstream>
00010 #include <assert.h>
00011 #include <string>
00012 #include <algorithm>
00013 #include "node.h"
00014 #include "edge.h"
00015
00020 namespace Markov {
00021
00022     template <typename NodeStorageType>
00023     class Node;
00024
00025     template <typename NodeStorageType>
00026     class Edge;
00027
00028     template <typename NodeStorageType>
00029
00041     class Model {
00042     public:
00043
00049         Model<NodeStorageType>();
00050
00082         NodeStorageType* RandomWalk(Markov::Random::RandomEngine* randomEngine, int minSetting, int
      maxSetting, NodeStorageType* buffer);
00083
00105         void AdjustEdge(const NodeStorageType* payload, long int occurrence);
00106
00122         bool Import(std::ifstream*);
00123
00133         bool Import(const char* filename);
00134
00151         bool Export(std::ofstream*);
00152
```

```
        bool Export(const char* filename);

        Node<NodeStorageType>* StarterNode(){ return starterNode;}

        std::vector<Edge<NodeStorageType>*>* Edges(){ return &edges;}

        std::map<NodeStorageType, Node<NodeStorageType>*>* Nodes(){ return &nodes;}

    private:
        std::map<NodeStorageType, Node<NodeStorageType>*> nodes;

        Node<NodeStorageType>* starterNode;


        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;
}




```

8.37 model_2gram.py File Reference

Namespaces

- model_2gram

Variables

- model_2gram.alphabet = string.printable

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

  output file handle

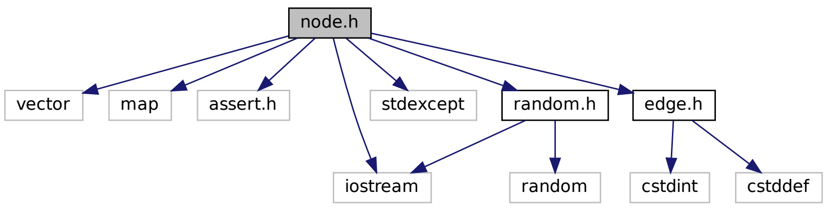
8.38 model_2gram.py

```
#!/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")
```

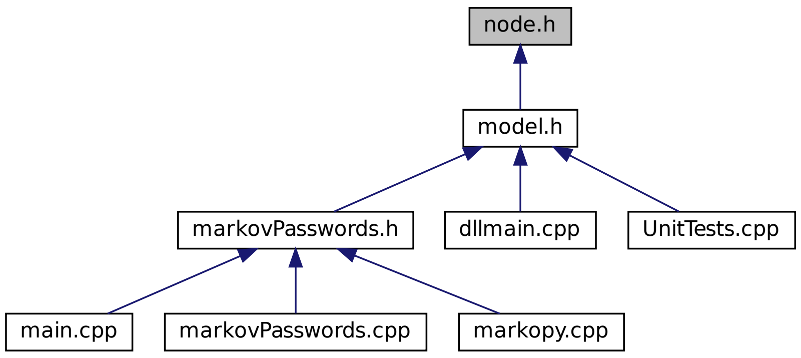
8.39 node.h File Reference

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

Include dependency graph for node.h:



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



Classes

- class Markov::Node< storageType >

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

Namespaces

- Markov

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

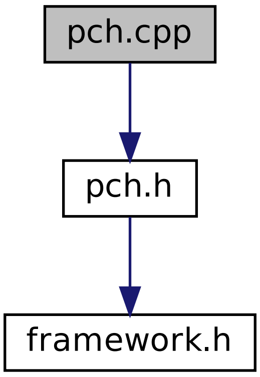
8.40 node.h

```
#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 {

    template <typename storageType>
    class Node {
    public:

        Node<storageType>();

        Node<storageType>(storageType _value);

        Edge<storageType>* Link(Node<storageType>*);

        Edge<storageType>* Link(Edge<storageType>*);

        Node<storageType>* RandomNext(Markov::Random::RandomEngine* randomEngine);

        bool UpdateEdges(Edge<storageType>*);

        Edge<storageType>* FindEdge(storageType repr);

        Edge<storageType>* FindEdge(Node<storageType>* target);

        inline unsigned char NodeValue();

        void UpdateTotalVerticeWeight(long int offset);

        inline std::map<storageType, Edge<storageType>*>* Edges();

        inline long int TotalEdgeWeights();


        std::vector<Edge<storageType>*> edgesV;
    private:


        storageType _value;
        long int total_edge_weights;
        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;
}

```

8.41 pch.cpp File Reference

`#include "pch.h"`

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



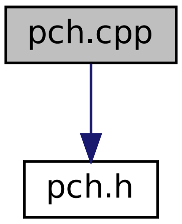
8.42 MarkovModel/src/pch.cpp

```
// 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.

```

8.43 pch.cpp File Reference

`#include "pch.h"`

Include dependency graph for UnitTests/pch.cpp:



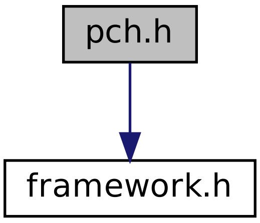
8.44 UnitTests/pch.cpp

```
// 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.

```

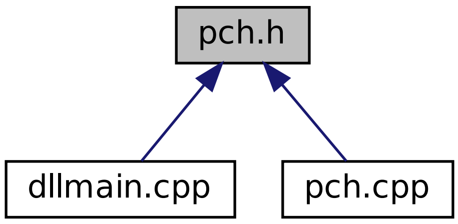
8.45 pch.h File Reference

`#include "framework.h"`

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



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

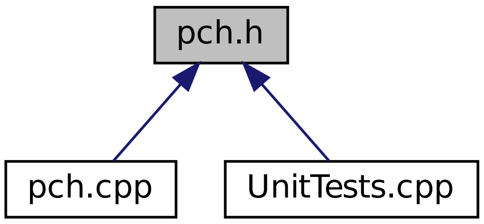


8.46 MarkovModel/src/pch.h

```
// 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

```

8.47 pch.h File Reference

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



8.48 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


```

8.49 random-model.py File Reference

Namespaces

- random-model
- random

Variables

- random-model.alphabet = string.printable

  password alphabet

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

  output file handle

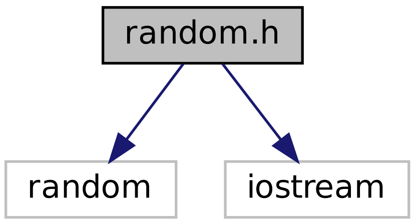
8.50 random-model.py

```
#!/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")
```

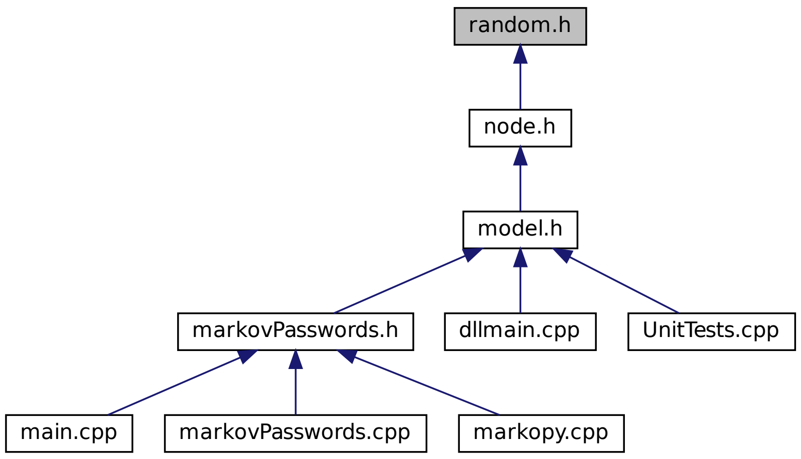
8.51 random.h File Reference

```
#include <random>
#include <iostream>
```

Include dependency graph for random.h:



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



Classes

- class Markov::Random::RandomEngine

    An abstract class for Random Engine.
- class Markov::Random::DefaultRandomEngine

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

    Implementation of Marsaglia Random Engine.
- class Markov::Random::Mersenne

    Implementation of Mersenne Twister Engine.

Namespaces

- Markov

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

    Objects related to RNG.

8.52 random.h

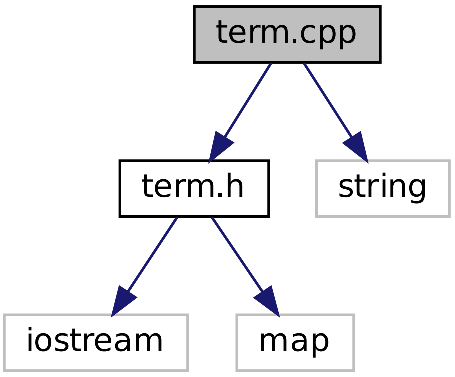
```


#include <random>
#include <iostream>

namespace Markov::Random{

    class RandomEngine{
    public:
        virtual inline unsigned long random() = 0;
    };



    class DefaultRandomEngine : public RandomEngine{
    public:
        inline unsigned long random(){
            return this->distribution()(this->generator());
        }
    protected:

        inline std::random_device& rd() {
            static std::random_device _rd;
            return _rd;
        }

        inline std::default_random_engine& generator() {
            static std::default_random_engine _generator(rd()());
            return _generator;
        }

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

    };


    class Marsaglia : public DefaultRandomEngine{
    public:

        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;
    };


    class Mersenne : public DefaultRandomEngine{

    };


};
```

8.53 README.md File Reference

8.54 term.cpp File Reference

```
#include "term.h"
#include <string>
```

Include dependency graph for term.cpp:



Functions

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

8.54.1 Function Documentation

8.54.1.1 operator<<()

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

overload for std::cout.

Definition at line 60 of file term.cpp.

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

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

8.55 term.cpp

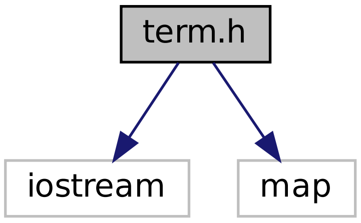
```
#pragma once
#include "term.h"
#include <string>

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

```

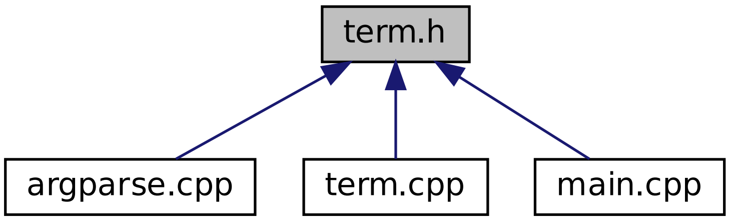
8.56 term.h File Reference

```
#include <iostream>
#include <map>
```

Include dependency graph for term.h:



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



Classes

- class Markov::API::CLI::Terminal

  pretty colors for Terminal. Windows Only.

Namespaces

- Markov

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

  Namespace for the MarkovPasswords API.
- Markov::API::CLI

  Structure to hold parsed cli 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)

8.56.1 Macro Definition Documentation

8.56.1.1 TERM_FAIL

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

Definition at line 10 of file term.h.

8.56.1.2 TERM_INFO

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

Definition at line 11 of file term.h.

8.56.1.3 TERM_SUCC

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

Definition at line 13 of file term.h.

8.56.1.4 TERM_WARN

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

Definition at line 12 of file term.h.

8.57 term.h

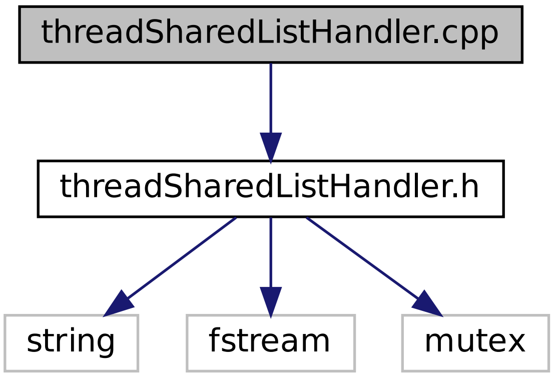
```
00001 #pragma once
00002
00003 #ifdef _WIN32
00004 #include <Windows.h>
00005 #endif
00006
00007 #include <iostream>
00008 #include <map>
00009
00010 #define TERM_FAIL "[" « Markov::API::CLI::Terminal::color::RED « "+" «
       Markov::API::CLI::Terminal::color::RESET « "] "
00011 #define TERM_INFO "[" « Markov::API::CLI::Terminal::color::BLUE « "+" «
       Markov::API::CLI::Terminal::color::RESET « "] "
00012 #define TERM_WARN "[" « Markov::API::CLI::Terminal::color::YELLOW « "+" «
       Markov::API::CLI::Terminal::color::RESET « "] "
00013 #define TERM_SUCC "[" « Markov::API::CLI::Terminal::color::GREEN « "+" «
       Markov::API::CLI::Terminal::color::RESET « "] "
00014
00015 namespace Markov::API::CLI{
00018     class Terminal {
00019     public:
00020
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
```

```



        static std::ostream endl;


    };

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

}

```

8.58 threadSharedListHandler.cpp File Reference

`#include "threadSharedListHandler.h"`

Include dependency graph for threadSharedListHandler.cpp:



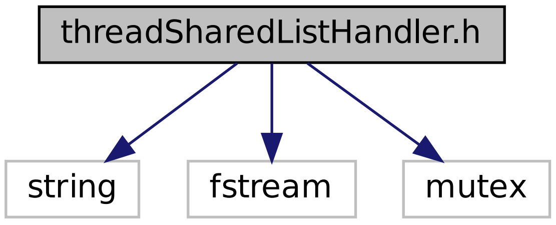
8.59 threadSharedListHandler.cpp

```
#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;
}
```

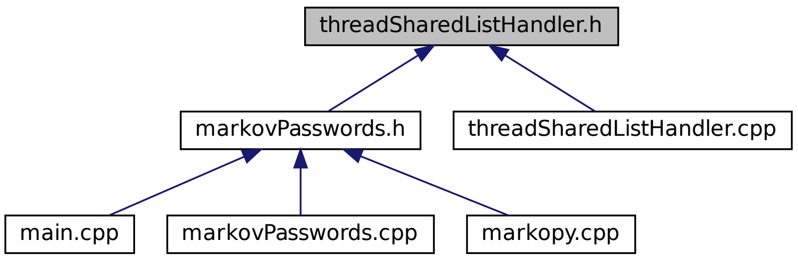
8.60 threadSharedListHandler.h File Reference

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

Include dependency graph for threadSharedListHandler.h:



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



Classes

- class Markov::API::Concurrency::ThreadSharedListHandler

    Simple class for managing shared access to file.

Namespaces

- Markov

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

    Namespace for the MarkovPasswords API.
- Markov::API::Concurrency

    Namespace for Concurrency related classes.

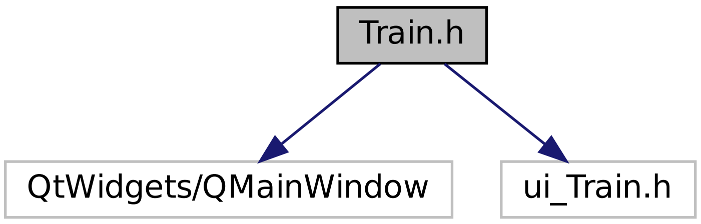
8.61 threadSharedListHandler.h

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

namespace Markov::API::Concurrency{

class ThreadSharedListHandler{
public:
    ThreadSharedListHandler(const char* filename);

    bool next(std::string* line);

private:
    std::ifstream listfile;
    std::mutex mlock;
};

};
```

8.62 Train.h File Reference

```
#include <QtWidgets/QMainWindow>
#include "ui_Train.h"
```

Include dependency graph for Train.h:



Classes

- class Markov::GUI::Train

  QT Training page class.

Namespaces

- Markov

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

  namespace for MarkovPasswords API GUI wrapper

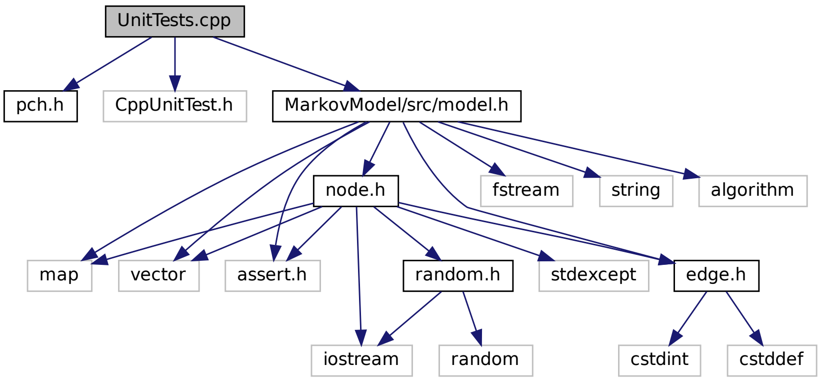
8.63 Train.h

```
#pragma once
#include <QtWidgets/QMainWindow>
#include "ui_Train.h"

namespace Markov::GUI{

    class Train :public QMainWindow {
    Q_OBJECT
    public:
        Train(QWidget* parent = Q_NULLPTR);

    private:
        Ui::Train ui;

    public slots:
        void home();
        void train();
    };
};

```

8.64 UnitTests.cpp File Reference

```
#include "pch.h"
#include "CppUnitTest.h"
#include "MarkovModel/src/model.h"
```

Include dependency graph for UnitTests.cpp:



Namespaces

- Testing

  Namespace for Microsoft Native Unit Testing Classes.
- Testing::MVP

  Testing Namespace for Minimal Viable Product.
- Testing::MVP::MarkovModel

  Testing Namespace for MVP MarkovModel.
- Testing::MVP::MarkovPasswords

  Testing namespace for MVP MarkovPasswords.
- Testing::MarkovModel

  Testing namespace for MarkovModel.
- Testing::MarkovPasswords

  Testing namespace for MarkovPasswords.

Functions

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

  Test class for minimal viable Edge.
- Testing::MVP::MarkovModel::TEST_CLASS (Node)

  Test class for minimal viable Node.
- Testing::MVP::MarkovModel::TEST_CLASS (Model)

  Test class for minimal viable Model.
- Testing::MVP::MarkovPasswords::TEST_CLASS (ArgParser)

  Test Class for Argparse class.
- Testing::MarkovModel::TEST_CLASS (Edge)

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

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

  Test class for rest of model cases.

8.65 UnitTests.cpp

```
#include "pch.h"
#include "CppUnitTest.h"
#include "MarkovModel/src/model.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;


namespace Testing {

    namespace MVP {
        namespace MarkovModel
        {
            TEST_CLASS(Edge)
            {
            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;
            }
    };

    TEST_CLASS(Node)
    {
    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;

                }
            };

            TEST_CLASS(Model)
            {
            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));
                }
            };
        }

        namespace MarkovPasswords
        {
            TEST_CLASS(ArgParser)
            {
            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"); */
                }



            };

        }
    }


    namespace MarkovModel {

        TEST_CLASS(Edge)
        {
        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);
            }
        };

        TEST_CLASS(Node)
        {
        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);
            }


        };

        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);

            }
        };

    }

    namespace MarkovPasswords {

    };

}
```

Index