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

0.7.0 Documentation

CHAPTER1

Markopy

Markopy

Generate wordlists with markov models.
HTML documentation · PDF documentation · Github Page · Report Bug · Add a Bug

Table of Contents

1.1 About The Project

This projects primary goal is to create a comfortable development environment for working with Markov Models, as well as creating an end product which can be used for generating password wordlists using Markov Models.
This project contains following sub-projects:

- MarkovModel
    - A versatile header-only template library for basic Markov Model structure.
- MarkovAPI
    - A static/dynamic library built on MarkovModel, specialized to generate single-word lines.
- MarkovAPICLI
    - A command line interface built on top of MarkovAPI
- Markopy
    - A CPython extension wrapper for MarkovAPI, along with its own command line interface.
- MarkovPasswordsGUI
    - A graphical user interface for MarkovAPI
- CudaMarkovAPI
    - GPU-accelerated wrapper for MarkovAPI
- CudaMarkopy
    - GPU-accelereted wrapper for CudaMarkovAPI

1.1.1 Possible Use Cases

While main focus of the development has been towards random walk performance and password generation, underlying libraries could be used for other applications such as specific use cases of hidden markov models in bioinformatics and gene research.

1.1.2 Getting Started

If you'd just like to use the project without contributing, check out the releases page. Latest minor release (0.8.x, 0.9.x) is even with main branch, and latest patch release (0.8.1, 0.8.2) is even with development branch.

1.1.3 Releases

Releases are maintained automatically via github actions. Each push to the main branch will trigger a minor version release, while each accepted pull request into the development branch will trigger a patch version release.
Pull requests to the development branch will also trigger a draft release only visible to the maintainers.
Release files contain:

- libmarkov-{version}-{platform}.zip
  - Depending on the platform, contains the libmarkov.so or markov.lib from that version.
- libcudamarkov-{version}-{platform}.zip
  - Depending on the platform, contains the libcudamarkov.so or cudamarkov.lib from that version.
- markopy-{version}-{platform}-py{ver}.{extension}.zip
  - Depending on the paltform, contains markopy.so or markopy.pyd. CPython extensions are compiled for specific versions. If your python version is not supported by the releases, you can create an issue, or build it yourself using the python3.x-dev package.
- cudamarkopy-{version}-{platform}-py{ver}.{extension}.zip
  - Depending on the paltform, contains cudamarkopy.so or cudamarkopy.pyd. CPython extensions are compiled for specific versions. If your python version is not supported by the releases, you can create an issue, or build it yourself using the python3.x-dev package.
- models-{version}.zip
  - Contains the latest models with the release version. Contains base models (untrained), trained models, and language-specific models.

1.2 Using the Project

You may use any section of this project, but we highly recommend using Markopy/CudaMarkopy python modules becaue they are optimized for the better user experience.

1.2.1 Using Markopy/CudaMarkopy

You can access basic operations from various model types using the python module, and if you are inexperienced with:

- Libmarkov and Libcudamarkov internals
- C++ code in general
- Importing and extending libraries in general
- Working with Python/C++ intermixed code

We strongly recommend using the python module.
While almost all of the python files provide their own entry point, you should use markopy.py or cudamarkopy.py depending on your preferences. Please note that CUDA code will not run without an NVIDIA graphics card, and without CUDA runtime.
markopy.py and cudamarkopy.py will let you select the model type you want to use with the -mt parameter. With each model, there are slightly different parameters available.
For top level CLI selector (markopy.py and cudamarkopy.py)

```
Model Mode selection choices:
usage: cudamarkopy.py [-mt MODEL_TYPE] [-h] [-ev EVALUATE] [-evt EVALUATE_TYPE]
Python wrapper for MarkovPasswords.
optional arguments:
  -mt MODEL_TYPE, --model_type MODEL_TYPE
                        Model type to use. Accepted values: MP, MMX
  -h, --help            Model type to use. Accepted values: MP, MMX
  -ev EVALUATE, --evaluate EVALUATE
```

```
                        Evaluate a models integrity
  -evt EVALUATE_TYPE, --evaluate_type EVALUATE_TYPE
                        Evaluation type, model or corpus
        Sample runs:
        markopy.py -mt MP generate trained.mdl -n 500 -w output.txt
            Import trained.mdl, and generate 500 lines to output.txt
        markopy.py -mt MMX generate trained.mdl -n 500 -w output.txt
            Import trained.mdl, and generate 500 lines to output.txt

        cudamarkopy.py -mt CUDA generate trained.mdl -n 500 -w output.txt
            Import trained.mdl, and generate 500 lines to output.txt
```

1.2.1.1 Help

You can use the `--help` function to print all the parameter details for all of the model types. Alternatively, you can combine it with `-mt` parameter to only print a single models parameters.

1.2.1.2 Evaluation

You can use the top level CLI selector to evaluate validity of a model or a corpus file (or many, if you provide a glob path pattern). A couple of examples:

Evaluate all of the models in the repository.

```
python3 cudamarkopy.py -ev "../../../models/**/*" -evt model
```

Example outputs:

- A well trained model

```
[+] Model: trained.mdl:
[+] total edges: 9024
[+] unique left nodes: 95
[+] unique right nodes: 95
################ Checks ################
[+] No dangling nodes             :✅
[+] Median in expected ratio      :✅
[+] Good bottom 10%               :✅
[+] 0 edges below threshold       :✅
[+] Model structure               :✅
[+] Model has any training        :✅
[+] Model has training            :✅
[+] Model training score: 3500088 :✅
```

- An untrained model:

```
[+] Model: 2gram.mdl:
[+] total edges: 9024
[+] unique left nodes: 95
[+] unique right nodes: 95
division by zero
[+] 0 weighted edges are dangerous and may halt the model.
[+] Model seems to be untrained
[+] Model is not adequately trained. Might result in inadequate results
################ Checks ################
[+] No dangling nodes             :✅
[+] Exceptionn in check_distrib   :❌
[+] Median in expected ratio      :✅
[+] Good bottom 10%               :✅
[+] Too many 0 edges              :❌
[+] Model structure               :✅
[+] Model has any training        :❌
[+] Model has training            :❌
[+] Model training score: 0.0     :❌
```

- A model with inadequate training due to small corpus file

```
[+] Model: corpus-Icelandic.mdl:
[+] total edges: 9024
[+] unique left nodes: 95
[+] unique right nodes: 95
[+] Model is not adequately trained. Might result in inadequate results
################ Checks ################
[+] No dangling nodes             :✅
[+] Median in expected ratio      :✅
[+] Good bottom 10%               :✅
[+] 0 edges below threshold       :✅
[+] Model structure               :✅
[+] Model has any training        :✅
[+] Model has training            :❌
[+] Model training score: 208.68  :❌
```

- A model with improper training due to alphabet conflicts (Mostly seen in languages with non-latin alphabets)

```
[+] Model: corpus-Japanese.mdl:
[+] total edges: 9024
[+] unique left nodes: 95
[+] unique right nodes: 95
[+] Median is too left leaning and might indicate high entropy
################ Checks ################
[+] No dangling nodes             :✅
[+] Median too left leaning       :❌
[+] Good bottom 10%               :✅
[+] 0 edges below threshold       :✅
[+] Model structure               :✅
[+] Model has any training        :✅
[+] Model has training            :✅
[+] Model training score: 10952   :✅
```

Evaluate a corpus file.

```
python3 cudamarkopy.py -ev "../../../datasets/pwdb.corpus"
[+] Corpus: graduation.corpus:
[+] Delimiter is: b'\t'
[+] Total number of lines: 157668136
[+] Sum of all string weights: 700089109
[+] Character total: 1498134485
[+] Average length: 9.501821503109545
[+] Average weight: 4.440270093635153
################ Checks ################
[+] No structural conflicts       :✅
```

1.2.1.3 Model selection

You may use the -mt parameter from markopy.py or cudamarkopy.py to select a model type. Allowed parameters are: MP, MMX, CUDA

Following, is each of the parameters required for these model types.

Following are applicable for -mt MP mode:

```
Python wrapper for MarkovPasswords.
positional arguments:
  mode                  Process mode. Either 'Train', 'Generate', or 'Combine'.
  input                 Input model file. This model will be imported before starting operation.
optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        Output model file. This model will be exported when done. Will be ignored for
       generation mode.
  -d DATASET, --dataset DATASET
                        Dataset file to read input from for training. Will be ignored for generation mode.
  -s SEPERATOR, --seperator SEPERATOR
                        Seperator character to use with training data.(character between occurrence and
       value)
  -t THREADS, --threads THREADS
                        Number of lines to generate. Ignored in training mode.
  -v, --verbosity       Output verbosity.
  -b, --bulk            Bulk generate or bulk train every corpus/model in the folder.
  -w WORDLIST, --wordlist WORDLIST
                        Wordlist file path to export generation results to. Will be ignored for training
       mode
  --min MIN             Minimum length that is allowed during generation
  --max MAX             Maximum length that is allowed during generation
  -n COUNT, --count COUNT
                        Number of lines to generate. Ignored in training mode.
Sample runs:
        base.py 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
        base.py generate trained.mdl -n 500 -w output.txt
            Import trained.mdl, and generate 500 lines to output.txt
        base.py 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.
        base.py 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.
usage: cudamarkopy.py [-h] [-o OUTPUT] [-d DATASET] [-s SEPERATOR] [-t THREADS] [-v] [-b] [-w WORDLIST]
                      [--min MIN] [--max MAX] [-n COUNT]
                      mode input
```

Following are applicable for -mt MMX mode:

```
usage: cudamarkopy.py [-h] [-t THREADS] [-v] [-b] [-w WORDLIST] [--min MIN] [--max MAX] [-n COUNT] [-st]
                      mode input
Python wrapper for MarkovPasswords.
positional arguments:
  mode                  Process mode. Either 'Train', 'Generate', or 'Combine'.
  input                 Input model file. This model will be imported before starting operation.
optional arguments:
  -h, --help            show this help message and exit
  -t THREADS, --threads THREADS
                        Number of lines to generate. Ignored in training mode.
```

```
  -v, --verbosity       Output verbosity.
  -b, --bulk            Bulk generate or bulk train every corpus/model in the folder.
  -w WORDLIST, --wordlist WORDLIST
                        Wordlist file path to export generation results to. Will be ignored for training
       mode
  --min MIN             Minimum length that is allowed during generation
  --max MAX             Maximum length that is allowed during generation
  -n COUNT, --count COUNT
                        Number of lines to generate. Ignored in training mode.
  -st, --stdout         Stdout mode
Sample runs:
        base.py 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
        base.py generate trained.mdl -n 500 -w output.txt
            Import trained.mdl, and generate 500 lines to output.txt
        base.py 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.
        base.py 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.
```

Following are applicable for -mt CUDA mode:

```
usage: cudamarkopy.py [-h] [-t THREADS] [-v] [-b] [-w WORDLIST] [--min MIN] [--max MAX] [-n COUNT] [-st]
       [-if]
                     mode input
Python wrapper for MarkovPasswords.
positional arguments:
  mode                  Process mode. Either 'Train', 'Generate', or 'Combine'.
  input                 Input model file. This model will be imported before starting operation.
optional arguments:
  -h, --help            show this help message and exit
  -t THREADS, --threads THREADS
                        Number of lines to generate. Ignored in training mode.
  -v, --verbosity       Output verbosity.
  -b, --bulk            Bulk generate or bulk train every corpus/model in the folder.
  -w WORDLIST, --wordlist WORDLIST
                        Wordlist file path to export generation results to. Will be ignored for training
       mode
  --min MIN             Minimum length that is allowed during generation
  --max MAX             Maximum length that is allowed during generation
  -n COUNT, --count COUNT
                        Number of lines to generate. Ignored in training mode.
  -st, --stdout         Stdout mode
  -if, --infinite       Infinite generation mode
Sample runs:
        base.py 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
        base.py generate trained.mdl -n 500 -w output.txt
            Import trained.mdl, and generate 500 lines to output.txt
        base.py 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.
        base.py 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.
```

1.2.1.4 Training

If you do not have a custom corpus to train with, you may use one of the pre-trained models from the github releases.

```
python3 cudamarkopy.py train ../../../models/base-models/2gram.mdl -d ../../../datasets/graduation.corpus -s
       "\\t" -o test.mdl -vvvvvvvvv
```

1.2.1.5 Generation

If you have not trained a model, you may download on of the trained models (preferably models/trained/trained.mdl) from the releases and use it for generation.

Generating to a file

```
python3 cudamarkopy.py generate trained.mdl -mt MMX -n 5000 --min 6 --max 12 -w test.txt -vvvv
```

Generating to stdout

```
python3 cudamarkopy.py generate trained.mdl -mt MMX -n 5000 --min 6 --max 12 --stdout
```

Generating with CUDA model WARNING Do not use CUDA model unless you want to generate 500M+ lines. Prefer stdout mode with pipes instead of writing to disk whenever possible

Using with Hashcat

```
python3 cudamarkopy.py generate trained.mdl -mt CUDA -n 5000 --min 6 --max 12 --stdout | hashcat -m 1400
       hashes.txt -O
```

Use with hashcat, continue until terminated

```
python3 cudamarkopy.py generate trained.mdl -mt CUDA -n 5000 --min 6 --max 12 --stdout --infinite | hashcat
       -m 1400 hashes.txt -O
```

1.3 Building

You can build the project using cmake with g++ & nvcc on linux, and msbuild & nvcc on windows.

1.3.1 Prerequisites

You can find a list of the dependencies below. If you have any missing, check out the setting up prequisites part.

1.3.1.1 General prequisites

To build the simple core of this project, you'll need:

1.3.1.1.1 Linux

- CMake, preferably one of the latest versions.
- CXX compiler, preferably g++ or clang++ (LLVM 3.9+).

1.3.1.1.2 Windows

- CMake, preferably one of the latest versions.
- CXX compiler, preferably msbuild(cl.exe) or clang++ (LLVM 3.9+). Please note that mingw is not recommended as it is not officially supported by the nvcc.exe, and might not be linkable if you are building the CUDA components too.

1.3.2 MarkovModel

This project does not have any extra dependencies, and it can be compiled with general dependencies without anything extra.

1.3.3 MarkovAPI

This project does not have any extra dependencies, and it can be compiled with general dependencies without anything extra.

1.3.4 MarkovAPICLI

- Boost.program_options (tested on 1.71.0-1.76.0)

1.3.5 Markopy

- Boost.Python (tested on 1.71.0-1.76.0)
- Python development package (tested on python 36-39)

1.3.6 CudaMarkovAPI

- CUDA toolkit (11.0+, c++17 support required)

1.3.7 CudaMarkopy

- CUDA toolkit (11.0+, c++17 support required)
- Boost.Python (tested on 1.71.0-1.76.0)
- Python development package (tested on python 36-39)

1.3.8 MarkovPasswordsGUI

- QT5 development environment. (qt5-qmake on ubuntu apt-get)
- QTWebEngine5 plugin. (qtwebengine5-dev on ubuntu apt-get)

1.3.9 CMake Configuration

You can build this project with cmake.
If you don't have prequisites for some of the projects set up, you can use the partial set up configuration to ignore those projects when setting the project up.
If you do not meet the prequisites, you'll have to partially set up the CMake file (you cant use –target to build some of the targets, because configuration phase will fail too).
Some examples for partially setting up and building the project are below.

1.3.9.1 Build everything

```
$ cmake . -DPYTHON_VER=38 && cmake --build .
```

This will build all the libraries and executables. Requires python-dev, CUDA, QT5, QT5-Webview

1.3.9.2 Build libraries only

```
$ cmake . -DPARTIAL=1 -DB_LIBS=1 && cmake --build .
```

Only build basic libraries. Requires only CXX compiler.

1.3.9.3 Build CUDA-accelerated libraries

```
$ cmake . -DPARTIAL=1 -DB_CUDA=1 && cmake --build .
```

Build libraries along with cuda accelerated ones.

1.3.9.4 Build python module & libraries

```
$ cmake . -DPARTIAL=1 -DPYTHON_VER=39 && cmake --build .
```

Will build basic libraries and python modules.

1.3.9.5 Build CUDA accelerated python module

```
$ cmake . -DPARTIAL=1 -DPYTHON_VER=39 -DB_CUDA=1 && cmake --build .
```

Will build cudamarkopy.

1.3.9.6 Build CUDA accelerated python module and the GUI

```
$ cmake . -DPARTIAL=1 -DPYTHON_VER=39 -DB_CUDA=1 -DB_GUI && cmake --build .
```

Combine methods

1.3.10 Installing Dependencies

1.3.10.0.1 Windows

- QT: Install QT For Windows
- Boost (program_options and python):
  - Download Boost from its website. Prefer one of the tested versions, 1.71.0 to 1.76.0
  - Unzip the contents.
  - Launch "Visual Studio Developer Command Prompt" (If you don't have this, properly set up the PATH% variable for cl.exe)
  - Move to the boost installation directory. Bootstrap libraries with your python version:
    ```
    .\bootstrap.bat --with-python=$(which python3.6) --with-python-version=3.6;
    ```
  - Run b2 to build the libraries.
    ```
    .\b2.exe --layout=system address-model=64 variant=release link=static runtime-link=shared
     threading=multi --with-program_options --with-python stage;
    ```
- Python: You can use the windows app store to download python runtime and libraries.

1.3.10.0.2 Linux

- QT: Follow this guide to install QT on Linux. Alternatively, on ubuntu you can `sudo apt-get install qt5-qmake qtwebengine5-dev`
- Boost (program options and python):
  - Download Boost from its website. Prefer one of the tested versions, 1.71.0 to 1.76.0
  - Unzip the contents.
  - Move to the boost installation directory. Bootstrap libraries with your python version:
    ```
    ./bootstrap.sh --with-python=$(which python3.6) --with-python-version=3.6;
    ```
  - Run `b2` to build the libraries.
    ```
    ./b2 variant=release link=static threading=multi --with-program_options install;
    ./b2 --with-python --buildid=3.6 install;
    ```
- Boost (alternative)
  - Use a package manager to install boost
    ```
    sudo apt-get install libboost-all-dev
    ```
- Python:
  ```
  sudo apt-get install python3-dev
  ```

1.4 File Structure

You may chose to create your own model structures or corpus files. Following, is the reference for the current structure for them.

1.4.1 Model

Model files are basically a list of edges in the model
Format is `{left_node_content},weight,{right_node_content}\n`
Example:
```
l,5,r
l,12,g
...
```
There are additional requirements for a model for entry and termination nodes. Entry nodes are represented with 0x00, and termination nodes are represented with 0xff
It is expected (but not mandatory) to have an edge from starting node to all of the other nodes, and same applies for edges from each nodes to the termination nodes.
If termination node has any edges that it is position on the left (meaning model file expectation is to traverse to another node after termination node) it will be loaded to the edges, but will be ignored during the random walk logic.

1.4.2 Corpus

Corpus files are used to train models. You may chose to train your own model (preferably over an existing base model, like 2gram.mdl) to have your own generation style.
Corpus file format is:
`{occurrence}{seperator}{string}\n`

- Occurrence is the weight associated with that string.
- Seperator is a single character seperator used to seperate occurrence with the string
- String is the password/string/sequence you want to add to the model.

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*` or `locate Python.h`.
If it doesn't, run `sudo apt-get install python3-dev`

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

Run:
```
ls /usr/lib/x86_64-linux-gnu/ | grep boost
```
and check the shared object filenames. A common issue is that lboost is required but filenames are formatted as libboost, or vice versa.
Do the same for python related library issues, run:
```
ls /usr/lib/x86_64-linux-gnu/ | grep python
```
to verify filename format is as required.
If not, you can modify the makefile, or create symlinks such as:
```
ln -s /usr/lib/x86_64-linux-gnu/libboost_python38.so /usr/lib/x86_64-linux-gnu/boost_python38.so
```

1.5.2 Windows

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

- Make sure you are working in the "Visual Studio Developer Command Prompt" terminal.
- Make sure you have Windows 10 SDK installed.
- From VS developer terminal, run echo INCLUDE%. If result does not have the windows sdk folders, run the following before running bootstrap (change your sdk version instead of 10.0.19041.0):
```
set INCLUDE=%INCLUDE%;C:\Program Files (x86)\Windows Kits\NETFXSDK\4.8\include\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.

1.6 Contributing

Feel free to contribute. We welcome all the issues and pull requests.

1.7 Contact

Twitter - @ahakcil

CHAPTER2

Deprecated List

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

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

CHAPTER3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

CHAPTER4

Hierarchical Index

4.1 Class Hierarchy

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

CHAPTER5

Class Index

5.1 Class List

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

CHAPTER6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

CHAPTER7

Namespace Documentation

7.1 base Namespace Reference

7.2 cudamarkopy Namespace Reference

Variables

- spec = spec_from_loader("markopy", SourceFileLoader("markopy", os.path.abspath("markopy.py")))
- markopy = module_from_spec(spec)
- mp = CudaMarkopyCLI()

7.2.1 Variable Documentation

7.2.1.1 markopy

```
cudamarkopy.markopy = module_from_spec(spec)
```
Definition at line 20 of file cudamarkopy.py.

7.2.1.2 mp

```
cudamarkopy.mp = CudaMarkopyCLI()
```
Definition at line 107 of file cudamarkopy.py.

7.2.1.3 spec

```
cudamarkopy.spec = spec_from_loader("markopy", SourceFileLoader("markopy", os.path.abspath("markopy.↩
py")))
```
Definition at line 19 of file cudamarkopy.py.

7.3 cudammx Namespace Reference

Variables

- string ext = "so"
- spec = spec_from_loader("cudamarkopy", ExtensionFileLoader("cudamarkopy", os.path.abspath(f"cudamarkopy.{ext}")))
- cudamarkopy = module_from_spec(spec)
- markopy = module_from_spec(spec)
- mp = CudaModelMatrixCLI()

7.3.1 Variable Documentation

7.3.1.1 cudamarkopy

```
cudammx.cudamarkopy = module_from_spec(spec)
```
Definition at line 20 of file cudammx.py.

7.3.1.2 ext

```
string cudammx.ext = "so"
```
Definition at line 15 of file cudammx.py.

7.3.1.3 markopy

```
cudammx.markopy = module_from_spec(spec)
```
Definition at line 35 of file cudammx.py.

7.3.1.4 mp

```
cudammx.mp = CudaModelMatrixCLI()
```
Definition at line 79 of file cudammx.py.

7.3.1.5 spec

```
cudammx.spec = spec_from_loader("cudamarkopy", ExtensionFileLoader("cudamarkopy", os.path.↩
abspath(f"cudamarkopy.{ext}")))
```
Definition at line 19 of file cudammx.py.

7.4 evaluate Namespace Reference

7.5 importer Namespace Reference

Functions

- def import_markopy ()

  import and return markopy module

7.5.1 Function Documentation

7.5.1.1 import_markopy()

```
def importer.import_markopy ( )
```
import and return markopy module

Returns

markopy module

Definition at line 12 of file importer.py.

```
00012 def import_markopy():
00013     """! @brief import and return markopy module
00014          @returns markopy module
00015     """
00016     ext = "so"
00017     if os.name == 'nt':
00018         ext="pyd"
00019     try:
00020         spec = spec_from_loader("markopy", ExtensionFileLoader("markopy",
      os.path.abspath(f"markopy.{ext}")))
00021         markopy = module_from_spec(spec)
00022         spec.loader.exec_module(markopy)
00023         return markopy
00024     except ImportError as e:
00025         #print(f"({__file__}) Working in development mode. Trying to load markopy.{ext} from
      ../../../out/")
00026         if(os.path.exists(f"../../../out/lib/markopy.{ext}")):
00027             spec = spec_from_loader("markopy", ExtensionFileLoader("markopy",
      os.path.abspath(f"../../../out/lib/markopy.{ext}")))
```

```
00028             markopy = module_from_spec(spec)
00029             spec.loader.exec_module(markopy)
00030             return markopy
00031         else:
00032             raise e
```

7.6 markopy Namespace Reference

Variables

- string ext = "so"
- markopy = import_markopy()
- mp = MarkopyCLI()

7.6.1 Variable Documentation

7.6.1.1 ext

```
string markopy.ext = "so"
```

Definition at line 24 of file markopy.py.

7.6.1.2 markopy

```
markopy.markopy = import_markopy()
```

Definition at line 35 of file markopy.py.

7.6.1.3 mp

```
markopy.mp = MarkopyCLI()
```

Definition at line 179 of file markopy.py.

7.7 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

  CPython module for Markov::API objects.
- Random

  Objects related to RNG.

Classes

- class Node

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

7.7.1 Detailed Description

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

7.8 Markov::API Namespace Reference

Namespace for the MarkovPasswords API.

Namespaces

- CLI

  Structure to hold parsed cli arguements.
- Concurrency

  Namespace for Concurrency related classes.
- CUDA

  Namespace for objects requiring CUDA libraries.

Classes

- class MarkovPasswords

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

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

7.8.1 Detailed Description

Namespace for the MarkovPasswords API.

7.9 Markov::API::CLI Namespace Reference

Structure to hold parsed cli arguements.

Classes

- struct _programOptions

  Structure to hold parsed cli arguements.
- class Argparse

  Parse command line arguements.
- class Terminal

  pretty colors for Terminal. Windows Only.

Typedefs

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

  Structure to hold parsed cli arguements.

Functions

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

7.9.1 Detailed Description

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

7.9.2 Typedef Documentation

7.9.2.1 ProgramOptions

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

7.9.3 Function Documentation

7.9.3.1 operator<<()

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

7.10 Markov::API::Concurrency Namespace Reference

Namespace for Concurrency related classes.

Classes

- class ThreadSharedListHandler

  Simple class for managing shared access to file.

7.10.1 Detailed Description

Namespace for Concurrency related classes.

7.11 Markov::API::CUDA Namespace Reference

Namespace for objects requiring CUDA libraries.

Namespaces

- Random

  *Namespace for Random engines operable under **device** space.*

Classes

- class CUDADeviceController

  Controller class for CUDA device.
- class CUDAModelMatrix

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

Functions

- __global__ void FastRandomWalkCUDAKernel (unsigned long int n, int minLen, int maxLen, char ∗output↩ Buffer, char ∗matrixIndex, long int ∗totalEdgeWeights, long int ∗valueMatrix, char ∗edgeMatrix, int matrixSize, int memoryPerKernelGrid, unsigned long ∗seed)

  CUDA kernel for the FastRandomWalk operation.
- __device__ char ∗ strchr (char ∗p, char c, int s_len)

  *srtchr implementation on **device** space*

7.11.1 Detailed Description

Namespace for objects requiring CUDA libraries.

7.11.2 Function Documentation

7.11.2.1 FastRandomWalkCUDAKernel()

```
__global__ void Markov::API::CUDA::FastRandomWalkCUDAKernel (
            unsigned long int n,
            int minLen,
            int maxLen,
            char * outputBuffer,
            char * matrixIndex,
            long int * totalEdgeWeights,
            long int * valueMatrix,
            char * edgeMatrix,
            int matrixSize,
            int memoryPerKernelGrid,
            unsigned long * seed )
```

CUDA kernel for the FastRandomWalk operation.
Will be initiated by CPU and continued by GPU (**global** tag)

Parameters

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

Definition at line 194 of file cudaModelMatrix.cu.

```
00195                                                       {
00196
00197         int kernelWorkerIndex = threadIdx.x + blockIdx.x * blockDim.x;
00198
00199         if(n==0) return;
00200
00201         char* e;
00202         int index = 0;
00203         char next;
00204         int len=0;
00205         long int selection;
00206         char cur;
00207         long int bufferctr = 0;
00208         unsigned long int *x,*y,*z,t;
00209         char* res = &outputBuffer[kernelWorkerIndex*memoryPerKernelGrid];
00210         x=&seed[kernelWorkerIndex*3];
00211         y=&seed[kernelWorkerIndex*3+1];
00212         z=&seed[kernelWorkerIndex*3+2];
00213         for (int i = 0; i < n; i++) {
00214             cur=199;
00215             len=0;
00216             while (true) {
00217                 e = strchr(matrixIndex, cur, matrixSize);
00218                 index = e - matrixIndex;
```

```
00219                 /*selection = Markov::API::CUDA::Random::devrandom(
00220                     seed[kernelWorkerIndex*3],
00221                     seed[kernelWorkerIndex*3+1],
00222                     seed[kernelWorkerIndex*3+2]) % totalEdgeWeights[index];*/
00223                 *x ^= *x « 16;
00224                 *x ^= *x » 5;
00225                 *x ^= *x « 1;
00226 
00227                 t = *x;
00228                 *x = *y;
00229                 *y = *z;
00230                 *z = t ^ *x ^ *y;
00231                 selection = *z % totalEdgeWeights[index];
00232             for(int j=0;j<matrixSize-1;j++){
00233                     selection -= valueMatrix[index*matrixSize + j];
00234                     if (selection < 0){
00235                         next = edgeMatrix[index*sizeof(char)*matrixSize + j];
00236                         break;
00237                     }
00238                 }
00239 
00240                 if (len >= maxLen)  break;
00241                 else if ((next < 0) && (len < minLen)) continue;
00242                 else if (next < 0) break;
00243                 cur = next;
00244                 res[bufferctr + len++] = cur;
00245             }
00246             res[bufferctr + len++] = '\n';
00247             bufferctr+=len;
00248         }
00249         res[bufferctr] = '\0';
00250     }
```

7.11.2.2 strchr()

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

srtchr implementation on **device** space
Fint the first matching index of a string

Parameters

p	- string to check
c	- character to match
s_len	- maximum string length

Returns

pointer to the match

Definition at line 252 of file cudaModelMatrix.cu.

```
00252                                                       {
00253         for (;; ++p, s_len--) {
00254             if (*p ==  c)
00255                 return((char *)p);
00256             if (!*p)
00257                 return((char *)NULL);
00258         }
00259     }
```

7.12 Markov::API::CUDA::Random Namespace Reference

Namespace for Random engines operable under **device** space.

Classes

- class Marsaglia

  *Extension of Markov::Random::Marsaglia which is capable o working on **device** space.*

Functions

- __device__ unsigned long devrandom (unsigned long &x, unsigned long &y, unsigned long &z)

    *Marsaglia Random Generation function operable in **device** space.*

7.12.1 Detailed Description

Namespace for Random engines operable under **device** space.

7.12.2 Function Documentation

7.12.2.1 devrandom()

```
__device__ unsigned long Markov::API::CUDA::Random::devrandom (
                unsigned long & x,
                unsigned long & y,
                unsigned long & z )
```

Marsaglia Random Generation function operable in **device** space.

Parameters

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

Returns

returns z

Definition at line 54 of file cudarandom.h.

```
00054                                                                                                   {
00055         unsigned long t;
00056         x ^= x « 16;
00057         x ^= x » 5;
00058         x ^= x « 1;
00059 
00060         t = x;
00061         x = y;
00062         y = z;
00063         z = t ^ x ^ y;
00064 
00065         return z;
00066     }
```

7.13 Markov::GUI Namespace Reference

namespace for MarkovPasswords API GUI wrapper

Classes

- class about

    QT Class for about page.

- class CLI

    QT CLI Class.

- class Train

    QT Training page class.

- class Generate

    QT Generation page class.

- class MarkovPasswordsGUI

  Reporting UI.
- class menu

  QT Menu class.

7.13.1 Detailed Description

namespace for MarkovPasswords API GUI wrapper

7.14 Markov::Markopy Namespace Reference

CPython module for Markov::API objects.

Namespaces

- CUDA

  CPython module for Markov::API::CUDA objects.

Functions

- BOOST_PYTHON_MODULE (markopy)

7.14.1 Detailed Description

CPython module for Markov::API objects.

7.14.2 Function Documentation

7.14.2.1 BOOST_PYTHON_MODULE()

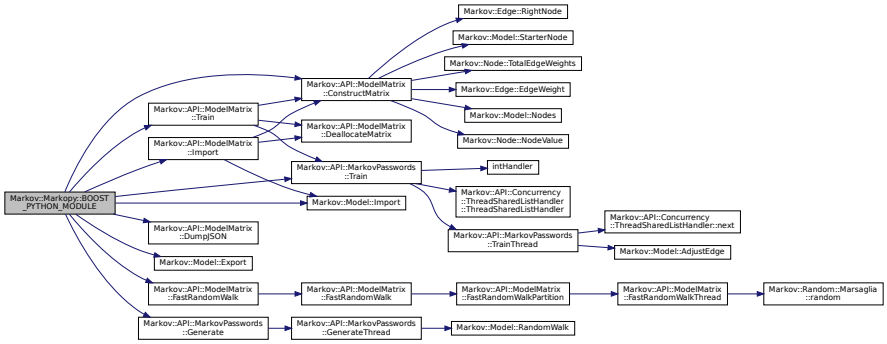
```
Markov::Markopy::BOOST_PYTHON_MODULE (
            markopy  )
```

Definition at line 37 of file markopy.cpp.

```
00038     {
00039         bool (Markov::API::MarkovPasswords::*Import)(const char*) = &Markov::Model<char>::Import;
00040         bool (Markov::API::MarkovPasswords::*Export)(const char*) = &Markov::Model<char>::Export;
00041         class_<Markov::API::MarkovPasswords>("MarkovPasswords", init<>())
00042             .def(init<>())
00043             .def("Train", &Markov::API::MarkovPasswords::Train)
00044             .def("Generate", &Markov::API::MarkovPasswords::Generate)
00045             .def("Import", Import, "Import a model file.")
00046             .def("Export", Export, "Export a model to file.")
00047         ;
00048
00049         int (Markov::API::ModelMatrix::*FastRandomWalk)(unsigned long int, const char*, int, int, int,
      bool)
00050             = &Markov::API::ModelMatrix::FastRandomWalk;
00051         class_<Markov::API::ModelMatrix>("ModelMatrix", init<>())
00052
00053             .def(init<>())
00054             .def("Train", &Markov::API::ModelMatrix::Train)
00055             .def("Import", &Markov::API::ModelMatrix::Import, "Import a model file.")
00056             .def("Export", Export, "Export a model to file.")
00057             .def("ConstructMatrix",&Markov::API::ModelMatrix::ConstructMatrix)
00058             .def("DumpJSON",&Markov::API::ModelMatrix::DumpJSON)
00059             .def("FastRandomWalk",FastRandomWalk)
00060             ;
00061     };
```

References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DumpJSON(), Markov::Model< NodeStorageTy
Markov::API::ModelMatrix::FastRandomWalk(), Markov::API::MarkovPasswords::Generate(), Markov::API::ModelMatrix::Import(),
Markov::Model< NodeStorageType >::Import(), Markov::API::MarkovPasswords::Train(), and Markov::API::ModelMatrix::Train().

Here is the call graph for this function:



7.15 Markov::Markopy::CUDA Namespace Reference

CPython module for Markov::API::CUDA objects.

Functions

- BOOST_PYTHON_MODULE (cudamarkopy)

7.15.1 Detailed Description

CPython module for Markov::API::CUDA objects.

7.15.2 Function Documentation

7.15.2.1 BOOST_PYTHON_MODULE()

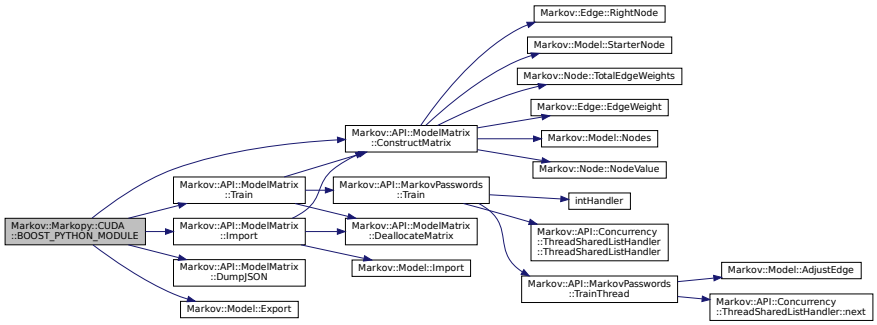
```
Markov::Markopy::CUDA::BOOST_PYTHON_MODULE (
            cudamarkopy  )
```

Definition at line 20 of file cudaMarkopy.cu.

```
00021     {
00022         bool (Markov::API::MarkovPasswords::*Export)(const char*) = &Markov::Model<char>::Export;
00023         void (Markov::API::CUDA::CUDAModelMatrix::*FastRandomWalk)(unsigned long int, const char*,
      int, int, bool, bool) = &Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk;
00024
00025         class_<Markov::API::CUDA::CUDAModelMatrix>("CUDAModelMatrix", init<>())
00026
00027             .def(init<>())
00028             .def("Train", &Markov::API::ModelMatrix::Train)
00029             .def("Import", &Markov::API::ModelMatrix::Import, "Import a model file.")
00030             .def("Export", Export, "Export a model to file.")
00031             .def("ConstructMatrix",&Markov::API::ModelMatrix::ConstructMatrix)
00032             .def("DumpJSON",&Markov::API::ModelMatrix::DumpJSON)
00033             .def("FastRandomWalk", FastRandomWalk)
00034             ;
00035     };
```

References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DumpJSON(), Markov::Model< NodeStorageTy
Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().

Here is the call graph for this function:



7.16 Markov::Random Namespace Reference

Objects related to RNG.

Classes

- class RandomEngine

  An abstract class for Random Engine.
- class DefaultRandomEngine

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

  Implementation of Marsaglia Random Engine.
- class Mersenne

  Implementation of Mersenne Twister Engine.

7.16.1 Detailed Description

Objects related to RNG.

7.17 mm Namespace Reference

Variables

- markopy = import_markopy()

7.17.1 Variable Documentation

7.17.1.1 markopy

```
mm.markopy = import_markopy()
```

Definition at line 11 of file mm.py.

7.18 mmx Namespace Reference

Variables

- markopy = import_markopy()
- mp = ModelMatrixCLI()

7.18.1 Variable Documentation

7.18.1.1 markopy

```
mmx.markopy = import_markopy()
```
Definition at line 12 of file mmx.py.

7.18.1.2 mp

```
mmx.mp = ModelMatrixCLI()
```
Definition at line 44 of file mmx.py.

7.19 model_2gram Namespace Reference

Variables

- alphabet = string.printable

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

  output file handle

7.19.1 Detailed Description

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

7.19.2 Variable Documentation

7.19.2.1 alphabet

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

7.19.2.2 f

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

7.20 mp Namespace Reference

Variables

- markopy = import_markopy()
- mp = MarkovPasswordsCLI()

7.20.1 Variable Documentation

7.20.1.1 markopy

```
mp.markopy = import_markopy()
```
Definition at line 13 of file mp.py.

7.20.1.2 mp

```
mp.mp = MarkovPasswordsCLI()
```
Definition at line 36 of file mp.py.

7.21 Python.CudaMarkopy Namespace Reference

wrapper scripts for CudaMarkopy

Classes

- class CudaMarkopyCLI

  CUDA extension to MarkopyCLI.
- class CudaModelMatrixCLI

  Python CLI wrapper for CudaModelMatrix.

7.21.1 Detailed Description

wrapper scripts for CudaMarkopy

7.22 Python.Markopy.Evaluation Namespace Reference

namespace for integrity evaluation

Classes

- class Evaluator

  Abstract class to evaluate and score integrity/validty.
- class ModelEvaluator

  evaluate a model
- class CorpusEvaluator

  evaluate a corpus

7.22.1 Detailed Description

namespace for integrity evaluation

7.23 random_model Namespace Reference

Variables

- alphabet = string.printable

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

  output file handle

7.23.1 Detailed Description

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

7.23.2 Variable Documentation

7.23.2.1 alphabet

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

7.23.2.2 f

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

7.24 Testing Namespace Reference

Namespace for Microsoft Native Unit Testing Classes.

Namespaces

- MarkovModel

  Testing namespace for MarkovModel.
- MarkovPasswords

  Testing namespace for MarkovPasswords.
- MVP

  Testing Namespace for Minimal Viable Product.

7.24.1 Detailed Description

Namespace for Microsoft Native Unit Testing Classes.

7.25 Testing::MarkovModel Namespace Reference

Testing namespace for MarkovModel.

Functions

- TEST_CLASS (Edge)

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

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

  Test class for rest of model cases.

7.25.1 Detailed Description

Testing namespace for MarkovModel.

7.25.2 Function Documentation

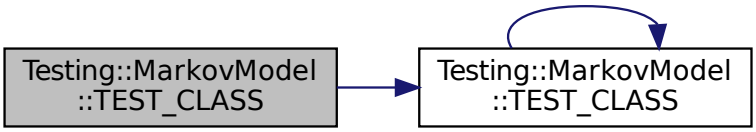
7.25.2.1 TEST_CLASS() [1/3]

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

Test class for rest of Edge cases.
send exception on integer underflow
test integer overflows
Definition at line 500 of file UnitTests.cpp.

```
00501         {
00502         public:
00505             TEST_METHOD(except_integer_underflow) {
00506                 auto _underflow_adjust = [] {
00507                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00508                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00509                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
      RightNode);
00510                     e->AdjustEdge(15);
00511                     e->AdjustEdge(-30);
00512                     delete LeftNode;
00513                     delete RightNode;
00514                     delete e;
00515                 };
00516                 Assert::ExpectException<std::underflow_error>(_underflow_adjust);
00517             }
00518 
00521             TEST_METHOD(except_integer_overflow) {
00522                 auto _overflow_adjust = [] {
00523                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00524                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00525                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
      RightNode);
00526                     e->AdjustEdge(~0ull);
00527                     e->AdjustEdge(1);
00528                     delete LeftNode;
00529                     delete RightNode;
00530                     delete e;
00531                 };
00532                 Assert::ExpectException<std::underflow_error>(_overflow_adjust);
00533             }
00534         };
```

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



7.25.2.2 TEST_CLASS() [2/3]

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

Test class for rest of model cases.
Definition at line 598 of file UnitTests.cpp.

```
00599         {
00600         public:
00601             TEST_METHOD(functional_random_walk) {
00602                 unsigned char* res2 = new unsigned char[12 + 5];
00603                 Markov::Random::Marsaglia MarsagliaRandomEngine;
00604                 Markov::Model<unsigned char> m;
00605                 Markov::Node<unsigned char>* starter = m.StarterNode();
00606                 Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00607                 Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
00608                 Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00609                 Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
```

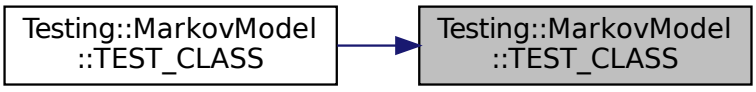
```
00610                 starter->Link(a)->AdjustEdge(1);
00611                 a->Link(b)->AdjustEdge(1);
00612                 b->Link(c)->AdjustEdge(1);
00613                 c->Link(end)->AdjustEdge(1);
00614 
00615                 char* res = (char*)m.RandomWalk(&MarsagliaRandomEngine,1,12,res2);
00616                 Assert::IsFalse(strcmp(res, "abc"));
00617             }
00618             TEST_METHOD(functionoal_random_walk_without_any) {
00619                 Markov::Model<unsigned char> m;
00620                 Markov::Node<unsigned char>* starter = m.StarterNode();
00621                 Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00622                 Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
00623                 Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00624                 Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
00625                 Markov::Edge<unsigned char>* res = NULL;
00626                 starter->Link(a)->AdjustEdge(1);
00627                 a->Link(b)->AdjustEdge(1);
00628                 b->Link(c)->AdjustEdge(1);
00629                 c->Link(end)->AdjustEdge(1);
00630 
00631                 res = starter->FindEdge('D');
00632                 Assert::IsNull(res);
00633 
00634             }
00635         };
```

References TEST_CLASS().

Referenced by TEST_CLASS().

Here is the call graph for this function:

Here is the caller graph for this function:



7.25.2.3 TEST_CLASS() [3/3]

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

Test class for rest of Node cases.
test RandomNext with 64 bit high values
test RandomNext with 64 bit high values
randomNext when no edges are present

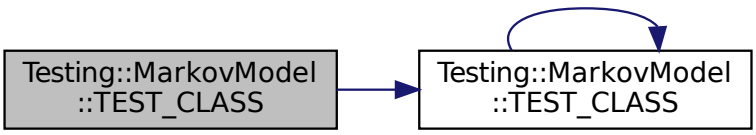
Definition at line 538 of file UnitTests.cpp.

```
00539         {
```

```
00540         public:
00541
00544             TEST_METHOD(rand_next_u64) {
00545                 Markov::Random::Marsaglia MarsagliaRandomEngine;
00546                 Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00547                 Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00548                 Markov::Edge<unsigned char>* e = src->Link(target1);
00549                 e->AdjustEdge((unsigned long)(1ull << 63));
00550                 Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00551                 Assert::IsTrue(res == target1);
00552                 delete src;
00553                 delete target1;
00554                 delete e;
00555
00556             }
00557
00560             TEST_METHOD(rand_next_u64_max) {
00561                 Markov::Random::Marsaglia MarsagliaRandomEngine;
00562                 Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00563                 Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00564                 Markov::Edge<unsigned char>* e = src->Link(target1);
00565                 e->AdjustEdge((0xffffFFFF));
00566                 Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00567                 Assert::IsTrue(res == target1);
00568                 delete src;
00569                 delete target1;
00570                 delete e;
00571
00572             }
00573
00576             TEST_METHOD(uninitialized_rand_next) {
00577
00578                 auto _invalid_next = [] {
00579                     Markov::Random::Marsaglia MarsagliaRandomEngine;
00580                     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00581                     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00582                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(src, target1);
00583                     Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00584
00585                     delete src;
00586                     delete target1;
00587                     delete e;
00588                 };
00589
00590                 Assert::ExpectException<std::logic_error>(_invalid_next);
00591             }
00592
00593
00594         };
```

References TEST_CLASS().

Here is the call graph for this function:



7.26 Testing::MarkovPasswords Namespace Reference

Testing namespace for MarkovPasswords.

7.26.1 Detailed Description

Testing namespace for MarkovPasswords.

7.27 Testing::MVP Namespace Reference

Testing Namespace for Minimal Viable Product.

Namespaces

- MarkovModel

  Testing Namespace for MVP MarkovModel.
- MarkovPasswords

  Testing namespace for MVP MarkovPasswords.

7.27.1 Detailed Description

Testing Namespace for Minimal Viable Product.

7.28 Testing::MVP::MarkovModel Namespace Reference

Testing Namespace for MVP MarkovModel.

Functions

- TEST_CLASS (Edge)

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

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

  Test class for minimal viable Model.

7.28.1 Detailed Description

Testing Namespace for MVP MarkovModel.

7.28.2 Function Documentation

7.28.2.1 TEST_CLASS() [1/3]

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

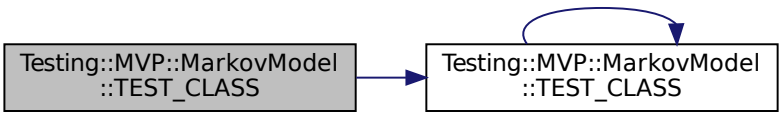
Test class for minimal viable Edge.
test default constructor
test linked constructor with two nodes
test AdjustEdge function
test TraverseNode returning RightNode
test LeftNode/RightNode setter
test negative adjustments
Definition at line 27 of file UnitTests.cpp.

```
00028             {
00029             public:
00030
00033                 TEST_METHOD(default_constructor) {
00034                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>;
00035                     Assert::IsNull(e->LeftNode());
00036                     Assert::IsNull(e->RightNode());
00037                     delete e;
00038                 }
00039
00042                 TEST_METHOD(linked_constructor) {
00043                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00044                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
```

```
00045                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
      RightNode);
00046                     Assert::IsTrue(LeftNode == e->LeftNode());
00047                     Assert::IsTrue(RightNode == e->RightNode());
00048                     delete LeftNode;
00049                     delete RightNode;
00050                     delete e;
00051                 }
00052
00055                 TEST_METHOD(AdjustEdge) {
00056                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00057                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00058                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
      RightNode);
00059                     e->AdjustEdge(15);
00060                     Assert::AreEqual(15ull, e->EdgeWeight());
00061                     e->AdjustEdge(15);
00062                     Assert::AreEqual(30ull, e->EdgeWeight());
00063                     delete LeftNode;
00064                     delete RightNode;
00065                     delete e;
00066                 }
00067
00070                 TEST_METHOD(TraverseNode) {
00071                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00072                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00073                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
      RightNode);
00074                     Assert::IsTrue(RightNode == e->TraverseNode());
00075                     delete LeftNode;
00076                     delete RightNode;
00077                     delete e;
00078                 }
00079
00082                 TEST_METHOD(set_left_and_right) {
00083                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00084                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00085                     Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(LeftNode,
      RightNode);
00086
00087                     Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>;
00088                     e2->SetLeftEdge(LeftNode);
00089                     e2->SetRightEdge(RightNode);
00090
00091                     Assert::IsTrue(e1->LeftNode() == e2->LeftNode());
00092                     Assert::IsTrue(e1->RightNode() == e2->RightNode());
00093                     delete LeftNode;
00094                     delete RightNode;
00095                     delete e1;
00096                     delete e2;
00097                 }
00098
00101                 TEST_METHOD(negative_adjust) {
00102                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00103                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00104                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
      RightNode);
00105                     e->AdjustEdge(15);
00106                     Assert::AreEqual(15ull, e->EdgeWeight());
00107                     e->AdjustEdge(-15);
00108                     Assert::AreEqual(0ull, e->EdgeWeight());
00109                     delete LeftNode;
00110                     delete RightNode;
00111                     delete e;
00112                 }
00113             };
```

References TEST_CLASS().

Here is the call graph for this function:



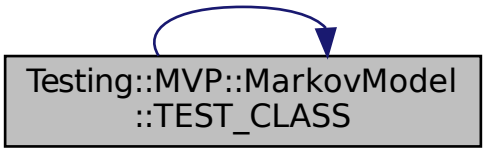
7.28.2.2 TEST_CLASS() [2/3]

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

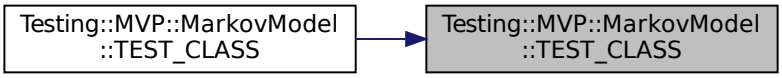
Test class for minimal viable Model.
test model constructor for starter node
test import
test export
test random walk
Definition at line 353 of file UnitTests.cpp.

```
00354             {
00355             public:
00358                 TEST_METHOD(model_constructor) {
00359                     Markov::Model<unsigned char> m;
00360                     Assert::AreEqual((unsigned char)'\0', m.StarterNode()->NodeValue());
00361                 }
00362
00365                 TEST_METHOD(import_filename) {
00366                     Markov::Model<unsigned char> m;
00367                     Assert::IsTrue(m.Import("../MarkovPasswords/Models/2gram.mdl"));
00368                 }
00369
00372                 TEST_METHOD(export_filename) {
00373                     Markov::Model<unsigned char> m;
00374                     Assert::IsTrue(m.Export("../MarkovPasswords/Models/testcase.mdl"));
00375                 }
00376
00379                 TEST_METHOD(random_walk) {
00380                     unsigned char* res = new unsigned char[12 + 5];
00381                     Markov::Random::Marsaglia MarsagliaRandomEngine;
00382                     Markov::Model<unsigned char> m;
00383                     Assert::IsTrue(m.Import("../Models/finished2.mdl"));
00384                     Assert::IsNotNull(m.RandomWalk(&MarsagliaRandomEngine,1,12,res));
00385                 }
00386             };
```

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



Here is the caller graph for this function:



7.28.2.3 TEST_CLASS() [3/3]

```
Testing::MVP::MarkovModel::TEST_CLASS (
```

Node)

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

Definition at line 117 of file UnitTests.cpp.

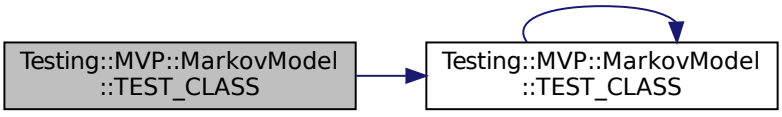
```
00118             {
00119             public:
00120
00123                 TEST_METHOD(default_constructor) {
00124                     Markov::Node<unsigned char>* n = new Markov::Node<unsigned char>();
00125                     Assert::AreEqual((unsigned char)0, n->NodeValue());
00126                     delete n;
00127                 }
00128
00131                 TEST_METHOD(uchar_constructor) {
00132                     Markov::Node<unsigned char>* n = NULL;
00133                     unsigned char test_cases[] = { 'c', 0x00, 0xff, -32 };
00134                     for (unsigned char tcase : test_cases) {
00135                         n = new Markov::Node<unsigned char>(tcase);
00136                         Assert::AreEqual(tcase, n->NodeValue());
00137                         delete n;
00138                     }
00139                 }
00140
00143                 TEST_METHOD(link_left) {
00144                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00145                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00146
00147                     Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
00148                     delete LeftNode;
00149                     delete RightNode;
00150                     delete e;
00151                 }
00152
00155                 TEST_METHOD(link_right) {
00156                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00157                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00158
00159                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(NULL, RightNode);
00160                     LeftNode->Link(e);
00161                     Assert::IsTrue(LeftNode == e->LeftNode());
00162                     Assert::IsTrue(RightNode == e->RightNode());
00163                     delete LeftNode;
00164                     delete RightNode;
00165                     delete e;
00166                 }
00167
00170                 TEST_METHOD(rand_next_low) {
00171                     Markov::Random::Marsaglia MarsagliaRandomEngine;
00172                     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00173                     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00174                     Markov::Edge<unsigned char>* e = src->Link(target1);
00175                     e->AdjustEdge(15);
00176                     Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00177                     Assert::IsTrue(res == target1);
00178                     delete src;
00179                     delete target1;
00180                     delete e;
00181
00182                 }
00183
00186                 TEST_METHOD(rand_next_u32) {
00187                     Markov::Random::Marsaglia MarsagliaRandomEngine;
00188                     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00189                     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00190                     Markov::Edge<unsigned char>* e = src->Link(target1);
00191                     e->AdjustEdge(1 << 31);
00192                     Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00193                     Assert::IsTrue(res == target1);
```

```
00194                   delete src;
00195                   delete target1;
00196                   delete e;
00197
00198               }
00199
00202               TEST_METHOD(rand_next_choice_1) {
00203                   Markov::Random::Marsaglia MarsagliaRandomEngine;
00204                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00205                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00206                   Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00207                   Markov::Edge<unsigned char>* e1 = src->Link(target1);
00208                   Markov::Edge<unsigned char>* e2 = src->Link(target2);
00209                   e1->AdjustEdge(1);
00210                   e2->AdjustEdge((unsigned long)(1ull << 31));
00211                   Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00212                   Assert::IsNotNull(res);
00213                   Assert::IsTrue(res == target2);
00214                   delete src;
00215                   delete target1;
00216                   delete e1;
00217                   delete e2;
00218               }
00219
00222               TEST_METHOD(rand_next_choice_2) {
00223                   Markov::Random::Marsaglia MarsagliaRandomEngine;
00224                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00225                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00226                   Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00227                   Markov::Edge<unsigned char>* e1 = src->Link(target1);
00228                   Markov::Edge<unsigned char>* e2 = src->Link(target2);
00229                   e2->AdjustEdge(1);
00230                   e1->AdjustEdge((unsigned long)(1ull << 31));
00231                   Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00232                   Assert::IsNotNull(res);
00233                   Assert::IsTrue(res == target1);
00234                   delete src;
00235                   delete target1;
00236                   delete e1;
00237                   delete e2;
00238               }
00239
00240
00243               TEST_METHOD(update_edges_count) {
00244
00245                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00246                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00247                   Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00248                   Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00249                   Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00250                   e1->AdjustEdge(25);
00251                   src->UpdateEdges(e1);
00252                   e2->AdjustEdge(30);
00253                   src->UpdateEdges(e2);
00254
00255                   Assert::AreEqual((size_t)2, src->Edges()->size());
00256
00257                   delete src;
00258                   delete target1;
00259                   delete e1;
00260                   delete e2;
00261
00262               }
00263
00266               TEST_METHOD(update_edges_total) {
00267
00268                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00269                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00270                   Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00271                   Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target1);
00272                   e1->AdjustEdge(25);
00273                   src->UpdateEdges(e1);
00274                   e2->AdjustEdge(30);
00275                   src->UpdateEdges(e2);
00276
00277                   //Assert::AreEqual(55ull, src->TotalEdgeWeights());
00278
00279                   delete src;
00280                   delete target1;
00281                   delete e1;
00282                   delete e2;
00283               }
00284
00285
00288               TEST_METHOD(find_vertice) {
00289
00290                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
```

```
00291                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00292                   Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00293                   Markov::Edge<unsigned char>* res = NULL;
00294                   src->Link(target1);
00295                   src->Link(target2);
00296
00297
00298                   res = src->FindEdge('b');
00299                   Assert::IsNotNull(res);
00300                   Assert::AreEqual((unsigned char)'b', res->TraverseNode()->NodeValue());
00301                   res = src->FindEdge('c');
00302                   Assert::IsNotNull(res);
00303                   Assert::AreEqual((unsigned char)'c', res->TraverseNode()->NodeValue());
00304
00305                   delete src;
00306                   delete target1;
00307                   delete target2;
00308
00309
00310               }
00311
00312
00315               TEST_METHOD(find_vertice_without_any) {
00316
00317                   auto _invalid_next = [] {
00318                       Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00319                       Markov::Edge<unsigned char>* res = NULL;
00320
00321                       res = src->FindEdge('b');
00322                       Assert::IsNull(res);
00323
00324                       delete src;
00325                   };
00326
00327                   //Assert::ExpectException<std::logic_error>(_invalid_next);
00328               }
00329
00332               TEST_METHOD(find_vertice_nonexistent) {
00333
00334                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00335                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00336                   Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00337                   Markov::Edge<unsigned char>* res = NULL;
00338                   src->Link(target1);
00339                   src->Link(target2);
00340
00341                   res = src->FindEdge('D');
00342                   Assert::IsNull(res);
00343
00344                   delete src;
00345                   delete target1;
00346                   delete target2;
00347
00348               }
00349           };
```

References TEST_CLASS().

Here is the call graph for this function:



7.29 Testing::MVP::MarkovPasswords Namespace Reference

Testing namespace for MVP MarkovPasswords.

Functions

- TEST_CLASS (ArgParser)

Test Class for Argparse class.

7.29.1 Detailed Description

Testing namespace for MVP MarkovPasswords.

7.29.2 Function Documentation

7.29.2.1 TEST_CLASS()

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

Test Class for Argparse class.
test basic generate
test basic generate reordered params
test basic generate param longnames
test basic generate
test basic train
test basic generate
Definition at line 395 of file UnitTests.cpp.

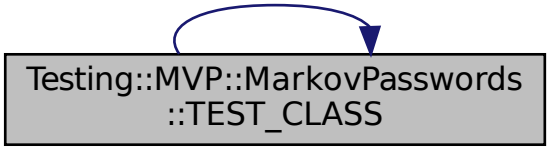
```
00396             {
00397             public:
00400                 TEST_METHOD(generate_basic) {
00401                     int argc = 8;
00402                     char *argv[] = {"markov.exe", "generate", "-if", "model.mdl", "-of",
      "passwords.txt", "-n", "100"};
00403
00404                     /*ProgramOptions *p = Argparse::parse(argc, argv);
00405                     Assert::IsNotNull(p);
00406
00407                     Assert::AreEqual(p->bImport, true);
00408                     Assert::AreEqual(p->bExport, false);
00409                     Assert::AreEqual(p->importname, "model.mdl");
00410                     Assert::AreEqual(p->outputfilename, "passwords.txt");
00411                     Assert::AreEqual(p->generateN, 100); */
00412
00413                 }
00414
00417                 TEST_METHOD(generate_basic_reorder) {
00418                     int argc = 8;
00419                     char *argv[] = { "markov.exe", "generate", "-n", "100", "-if", "model.mdl", "-of",
      "passwords.txt" };
00420
00421                     /*ProgramOptions* p = Argparse::parse(argc, argv);
00422                     Assert::IsNotNull(p);
00423
00424                     Assert::AreEqual(p->bImport, true);
00425                     Assert::AreEqual(p->bExport, false);
00426                     Assert::AreEqual(p->importname, "model.mdl");
00427                     Assert::AreEqual(p->outputfilename, "passwords.txt");
00428                     Assert::AreEqual(p->generateN, 100);*/
00429                 }
00430
00433                 TEST_METHOD(generate_basic_longname) {
00434                     int argc = 8;
00435                     char *argv[] = { "markov.exe", "generate", "-n", "100", "--inputfilename",
      "model.mdl", "--outputfilename", "passwords.txt" };
00436
00437                     /*ProgramOptions* p = Argparse::parse(argc, argv);
00438                     Assert::IsNotNull(p);
00439
00440                     Assert::AreEqual(p->bImport, true);
00441                     Assert::AreEqual(p->bExport, false);
00442                     Assert::AreEqual(p->importname, "model.mdl");
00443                     Assert::AreEqual(p->outputfilename, "passwords.txt");
00444                     Assert::AreEqual(p->generateN, 100); */
00445                 }
00446
00449                 TEST_METHOD(generate_fail_badmethod) {
00450                     int argc = 8;
00451                     char *argv[] = { "markov.exe", "junk", "-n", "100", "--inputfilename",
      "model.mdl", "--outputfilename", "passwords.txt" };
00452
00453                     /*ProgramOptions* p = Argparse::parse(argc, argv);
00454                     Assert::IsNull(p); */
```

```
00455                 }
00456
00459                 TEST_METHOD(train_basic) {
00460                     int argc = 4;
00461                     char *argv[] = { "markov.exe", "train", "-ef", "model.mdl" };
00462
00463                     /*ProgramOptions* p = Argparse::parse(argc, argv);
00464                     Assert::IsNotNull(p);
00465
00466                     Assert::AreEqual(p->bImport, false);
00467                     Assert::AreEqual(p->bExport, true);
00468                     Assert::AreEqual(p->exportname, "model.mdl"); */
00469
00470                 }
00471
00474                 TEST_METHOD(train_basic_longname) {
00475                     int argc = 4;
00476                     char *argv[] = { "markov.exe", "train", "--exportfilename", "model.mdl" };
00477
00478                     /*ProgramOptions* p = Argparse::parse(argc, argv);
00479                     Assert::IsNotNull(p);
00480
00481                     Assert::AreEqual(p->bImport, false);
00482                     Assert::AreEqual(p->bExport, true);
00483                     Assert::AreEqual(p->exportname, "model.mdl"); */
00484                 }
00485
00486
00487
00488             };
```

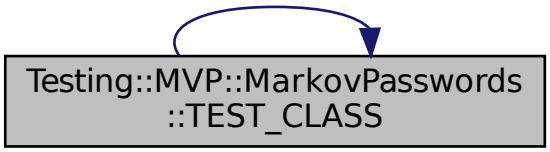
References TEST_CLASS().

Referenced by TEST_CLASS().

Here is the call graph for this function:



Here is the caller graph for this function:



CHAPTER8

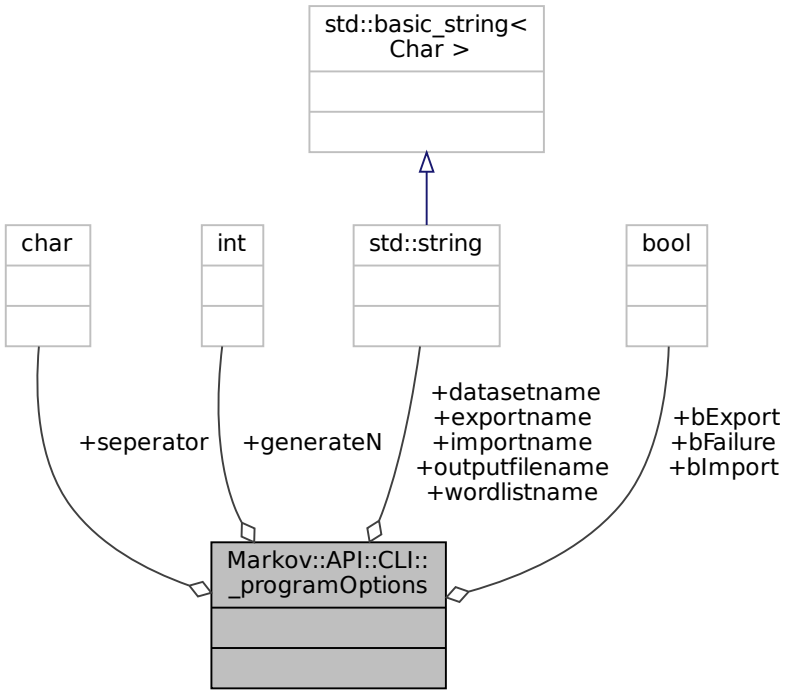
Class Documentation

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

Structure to hold parsed cli arguements.

`#include <argparse.h>`

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



Public Attributes

- bool bImport

  Import flag to validate import

- bool bExport

  Export flag to validate export

- bool bFailure

  Failure flag to validate succesfull running

- char seperator

  Seperator character to use with training data. (character between occurence and value)"

- std::string importname

Import name of our model.

- std::string exportname

  Import name of our given wordlist

- std::string wordlistname

  Import name of our given wordlist

- std::string outputfilename

  Output name of our generated password list

- std::string datasetname

  The name of the given dataset

- int generateN

  Number of passwords to be generated

8.1.1 Detailed Description

Structure to hold parsed cli arguements.
Definition at line 26 of file argparse.h.

8.1.2 Member Data Documentation

8.1.2.1 bExport

`bool Markov::API::CLI::_programOptions::bExport`
Export flag to validate export

Definition at line 35 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::Argparse(), and Markov::API::CLI::Argparse::setProgramOptions().

8.1.2.2 bFailure

`bool Markov::API::CLI::_programOptions::bFailure`
Failure flag to validate succesfull running

Definition at line 40 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::Argparse(), and Markov::API::CLI::Argparse::setProgramOptions().

8.1.2.3 bImport

`bool Markov::API::CLI::_programOptions::bImport`
Import flag to validate import

Definition at line 30 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::Argparse(), and Markov::API::CLI::Argparse::setProgramOptions().

8.1.2.4 datasetname

`std::string Markov::API::CLI::_programOptions::datasetname`
The name of the given dataset

Definition at line 70 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::Argparse(), and Markov::API::CLI::Argparse::setProgramOptions().

8.1.2.5 exportname

`std::string Markov::API::CLI::_programOptions::exportname`
Import name of our given wordlist

Definition at line 55 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::setProgramOptions().

8.1.2.6 generateN

`int Markov::API::CLI::_programOptions::generateN`
Number of passwords to be generated

Definition at line 75 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::Argparse(), and Markov::API::CLI::Argparse::setProgramOptions().

8.1.2.7 importname

`std::string Markov::API::CLI::_programOptions::importname`
Import name of our model.
Definition at line 50 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::Argparse(), and Markov::API::CLI::Argparse::setProgramOptions().

8.1.2.8 outputfilename

`std::string Markov::API::CLI::_programOptions::outputfilename`
Output name of our generated password list

Definition at line 65 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::Argparse(), and Markov::API::CLI::Argparse::setProgramOptions().

8.1.2.9 seperator

`char Markov::API::CLI::_programOptions::seperator`
Seperator character to use with training data. (character between occurence and value)"

Definition at line 45 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::setProgramOptions().

8.1.2.10 wordlistname

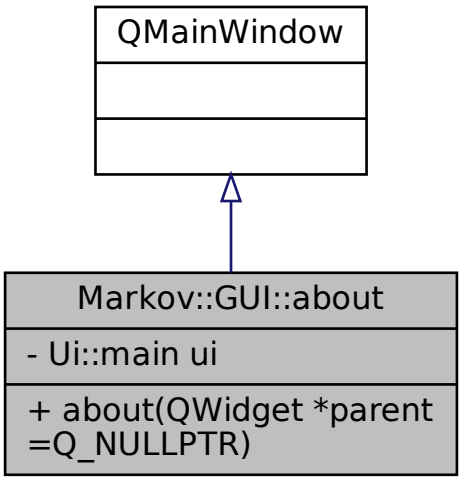
`std::string Markov::API::CLI::_programOptions::wordlistname`
Import name of our given wordlist

Definition at line 60 of file argparse.h.
Referenced by Markov::API::CLI::Argparse::Argparse().
The documentation for this struct was generated from the following file:

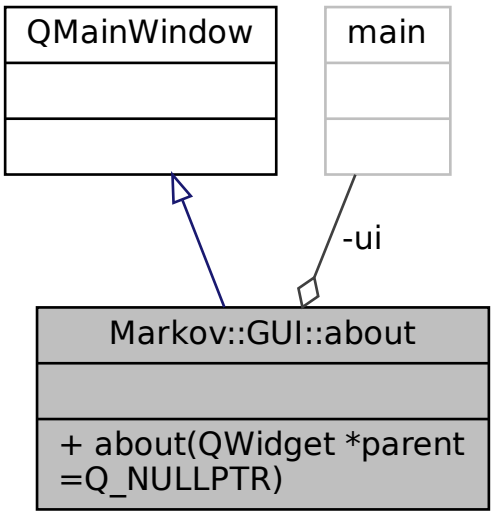
- Markopy/MarkovAPICLI/src/argparse.h

8.2 Markov::GUI::about Class Reference

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



Collaboration diagram for Markov::GUI::about:



Public Member Functions

- about (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::main ui

8.2.1 Detailed Description

QT Class for about page.
Definition at line 18 of file about.h.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 about()

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

Definition at line 14 of file about.cpp.

```
00015     : QMainWindow(parent)
00016 {
00017     ui.setupUi(this);
00018
00019 }
```

References ui.

8.2.3 Member Data Documentation

8.2.3.1 ui

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

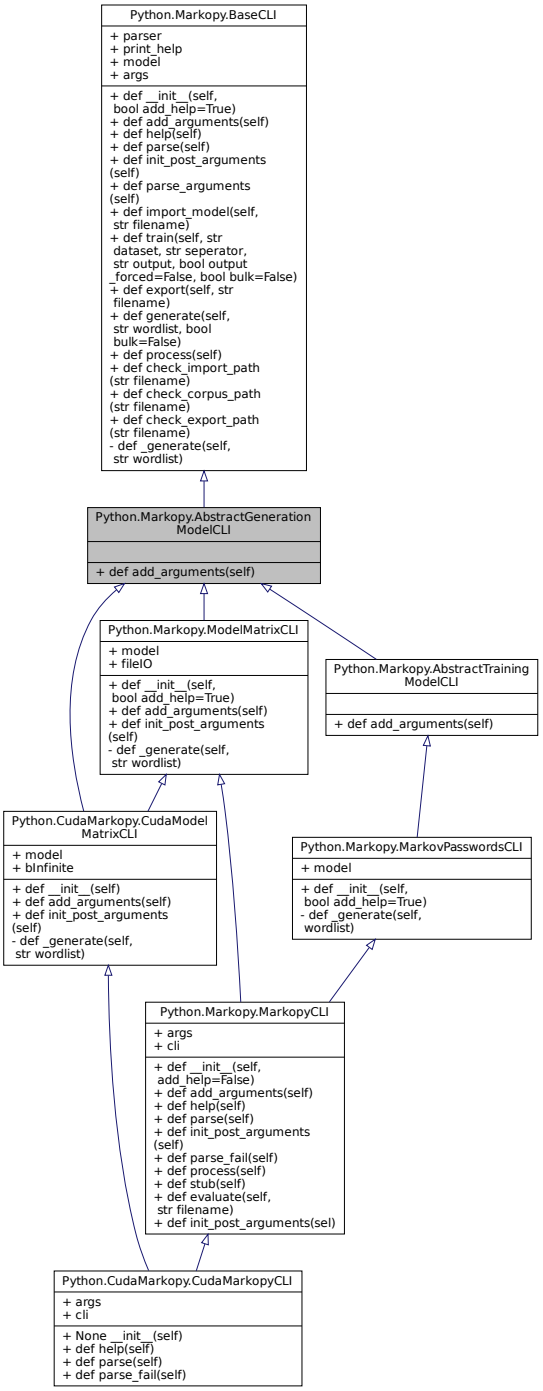
Definition at line 24 of file about.h.
Referenced by about().
The documentation for this class was generated from the following files:

- Markopy/MarkovPasswordsGUI/src/about.h
- Markopy/MarkovPasswordsGUI/src/about.cpp

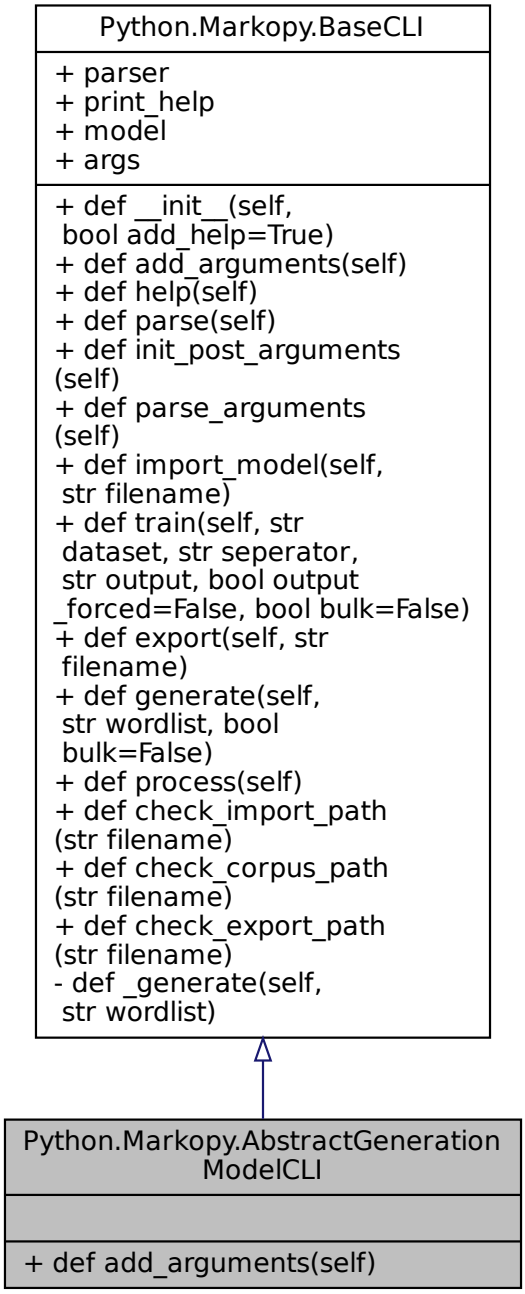
8.3 Python.Markopy.AbstractGenerationModelCLI Class Reference

abstract class for generation capable models

Inheritance diagram for Python.Markopy.AbstractGenerationModelCLI:



Collaboration diagram for Python.Markopy.AbstractGenerationModelCLI:



Public Member Functions

- def add_arguments (self)
- def help (self)
- def parse (self)
- def init_post_arguments (self)
- def parse_arguments (self)
- def import_model (self, str filename)

Import a model file.

- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def process (self)

  Process parameters for operation.

Static Public Member Functions

- def check_import_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity

Public Attributes

- parser
- print_help
- model
- args

Private Member Functions

- def _generate (self, str wordlist)

  wrapper for generate function.

8.3.1 Detailed Description

abstract class for generation capable models
Definition at line 257 of file base.py.

8.3.2 Member Function Documentation

8.3.2.1 _generate()

```
def Python.Markopy.BaseCLI._generate (
              self,
             str wordlist )  [private], [inherited]
```

wrapper for generate function.
This can be overloaded by other models

Parameters

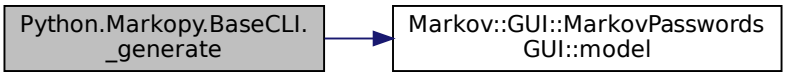
wordlist	filename to generate to

Reimplemented in Python.Markopy.ModelMatrixCLI, and Python.CudaMarkopy.CudaModelMatrixCLI.
Definition at line 161 of file base.py.

```
00161     def _generate(self, wordlist : str):
00162         """!
```

```
00163         @brief wrapper for generate function. This can be overloaded by other models
00164         @param wordlist filename to generate to
00165         """
00166         self.model.Generate(int(self.args.count), wordlist, int(self.args.min), int(self.args.max),
      int(self.args.threads))
00167
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.generate().
Here is the call graph for this function:



Here is the caller graph for this function:

8.3.2.2 add_arguments()

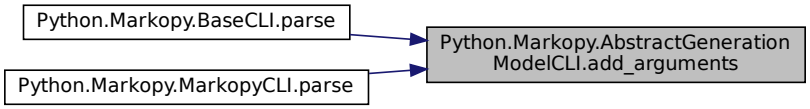
```
def Python.Markopy.AbstractGenerationModelCLI.add_arguments (
            self )
```

Reimplemented from Python.Markopy.BaseCLI.
Reimplemented in Python.Markopy.ModelMatrixCLI, Python.Markopy.MarkopyCLI, Python.Markopy.AbstractTrainingModelCLI, and Python.CudaMarkopy.CudaModelMatrixCLI.
Definition at line 264 of file base.py.

```
00264     def add_arguments(self):
00265         "Add command line arguements to the parser"
00266         super().add_arguments()
00267         self.parser.add_argument("input",                          help="Input model file. This
      model will be imported before starting operation.")
00268         self.parser.add_argument("-w", "--wordlist",               help="Wordlist file path to
      export generation results to. Will be ignored for training mode")
00269         self.parser.add_argument("--min", default=6,               help="Minimum length that is
      allowed during generation")
00270         self.parser.add_argument("--max", default=12,              help="Maximum length that is
      allowed during generation")
00271         self.parser.add_argument("-n", "--count",                  help="Number of lines to
      generate. Ignored in training mode.")
00272
00273
```

References Python.Markopy.BaseCLI.parser.
Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.3.2.3 check_corpus_path()

```
def Python.Markopy.BaseCLI.check_corpus_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186 
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.3.2.4 check_export_path()

```
def Python.Markopy.BaseCLI.check_export_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197 
00198         if(filename and os.path.isfile(filename)):
00199             return True
```

```
00200         return True
00201
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.3.2.5 check_import_path()

```
def Python.Markopy.BaseCLI.check_import_path (
             str filename )  [static], [inherited]
```

check import path for validity

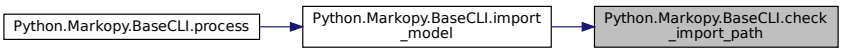
Parameters

filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:



8.3.2.6 export()

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

Parameters

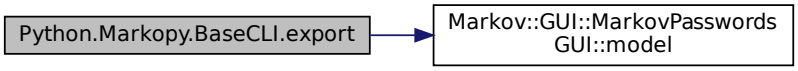
filename	filename to export to

Definition at line 138 of file base.py.

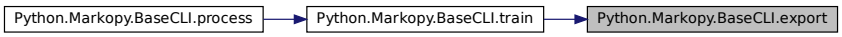
```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
```

```
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.3.2.7 generate()

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

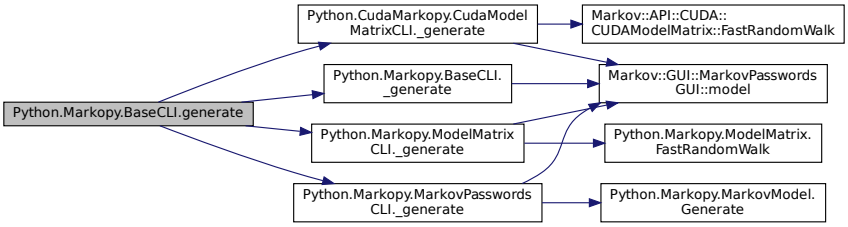
model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop
Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.
Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

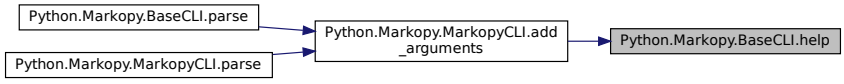
8.3.2.8 help()

```
def Python.Markopy.BaseCLI.help (
              self )  [inherited]
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.
Definition at line 51 of file base.py.

```
00051     def help(self):
00052         "! @brief Handle help strings. Defaults to argparse's help"
00053         self.print_help()
00054
```

References Python.Markopy.BaseCLI.print_help.
Referenced by Python.Markopy.MarkopyCLI.add_arguments().
Here is the caller graph for this function:



8.3.2.9 import_model()

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

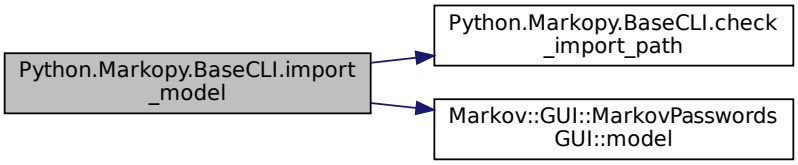
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

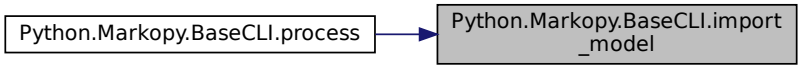
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.3.2.10 init_post_arguments()

```
def Python.Markopy.BaseCLI.init_post_arguments (
              self )  [inherited]
```

Reimplemented in Python.Markopy.ModelMatrixCLI, Python.Markopy.MarkopyCLI, Python.CudaMarkopy.CudaModelMatrixCLI, and Python.Markopy.MarkopyCLI.

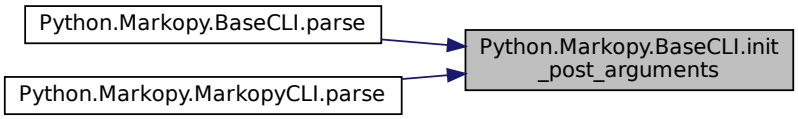
Definition at line 62 of file base.py.

```
00062     def init_post_arguments(self):
00063         "! @brief set up stuff that is collected from command line arguements"
00064         logging.VERBOSITY = 0
00065         try:
00066             if self.args.verbosity:
00067                 logging.VERBOSITY = self.args.verbosity
00068                 logging.pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00069         except:
00070             pass
00071 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.3.2.11 parse()

```
def Python.Markopy.BaseCLI.parse (
              self )  [inherited]
```

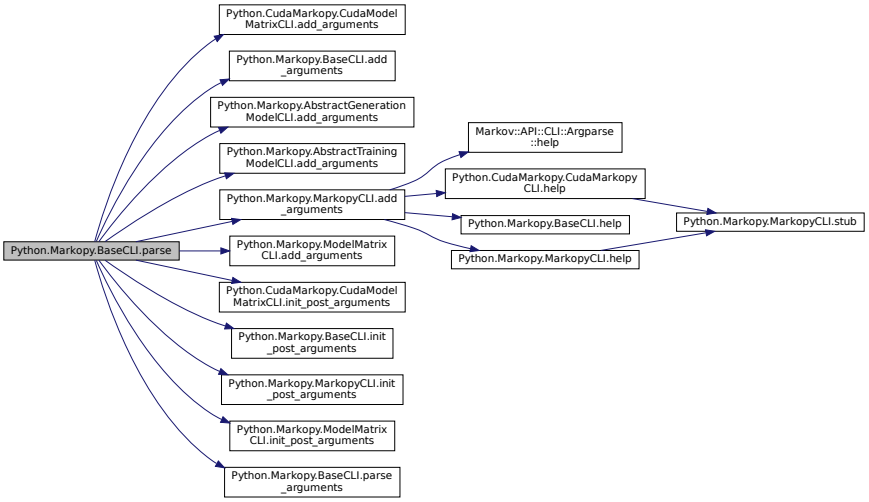
Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.

Definition at line 55 of file base.py.

```
00055     def parse(self):
00056         "! @brief add, parse and hook arguements"
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMode Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.init_post_arguments(), Python.Markopy.ModelMatrixC and Python.Markopy.BaseCLI.parse_arguments().

Here is the call graph for this function:



8.3.2.12 parse_arguments()

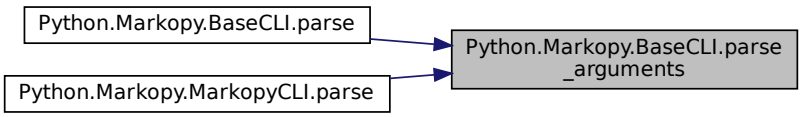
```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
```

```
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().
Here is the caller graph for this function:



8.3.2.13 process()

```
def Python.Markopy.BaseCLI.process (
              self )  [inherited]
```

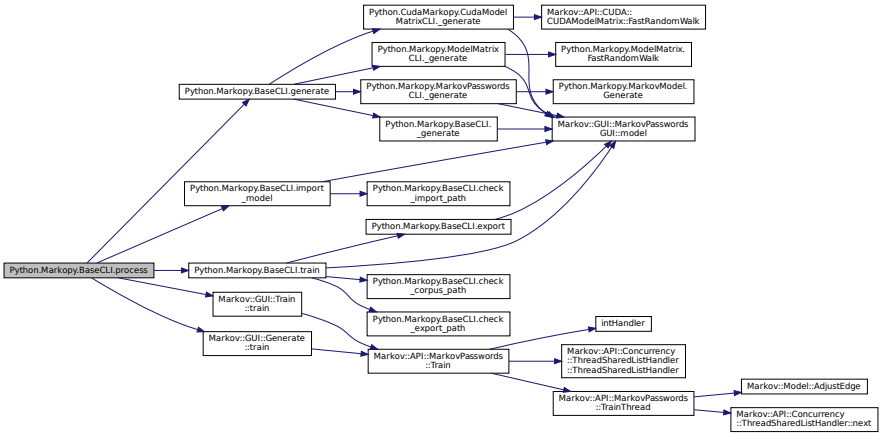
Process parameters for operation.
Reimplemented in Python.Markopy.MarkopyCLI.
Definition at line 202 of file base.py.

```
00202     def process(self):
00203         """!
00204         @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging.pprint(f"Bulk mode operation chosen.", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
      (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00210                     corpus_list = os.listdir(self.args.dataset)
00211                     for corpus in corpus_list:
00212                         self.import_model(self.args.input)
00213                         logging.pprint(f"Training {self.args.input} with {corpus}", 2)
00214                         output_file_name = corpus
00215                         model_extension = ""
00216                         if "." in self.args.input:
00217                             model_extension = self.args.input.split(".")[-1]
00218                         self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
      f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00219                 else:
00220                     logging.pprint("In bulk training, output and dataset should be a directory.")
00221                     exit(1)
00222
00223             elif (self.args.mode.lower() == "generate"):
00224                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
      (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00225                     model_list = os.listdir(self.args.input)
00226                     print(model_list)
00227                     for input in model_list:
00228                         logging.pprint(f"Generating from {self.args.input}/{input} to
      {self.args.wordlist}/{input}.txt", 2)
00229                         self.import_model(f"{self.args.input}/{input}")
00230                         model_base = input
00231                         if "." in self.args.input:
00232                             model_base = input.split(".")[1]
00233                         self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00234                 else:
00235                     logging.pprint("In bulk generation, input and wordlist should be directory.")
00236
00237         else:
00238             self.import_model(self.args.input)
00239             if (self.args.mode.lower() == "generate"):
00240                 self.generate(self.args.wordlist)
00241
00242
00243             elif (self.args.mode.lower() == "train"):
00244                 self.train(self.args.dataset, self.args.seperator, self.args.output,
      output_forced=True)
00245
00246
00247             elif(self.args.mode.lower() == "combine"):
00248                 self.train(self.args.dataset, self.args.seperator, self.args.output)
00249                 self.generate(self.args.wordlist)
```

```
00250
00251
00252             else:
00253                 logging.pprint("Invalid mode arguement given.")
00254                 logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255                 exit(5)
00256
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.generate(), Python.Markopy.BaseCLI.import_model(), Markov::GUI::Generate.train(), Markov::GUI::Train.train(), and Python.Markopy.BaseCLI.train().

Here is the call graph for this function:



8.3.2.14 train()

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

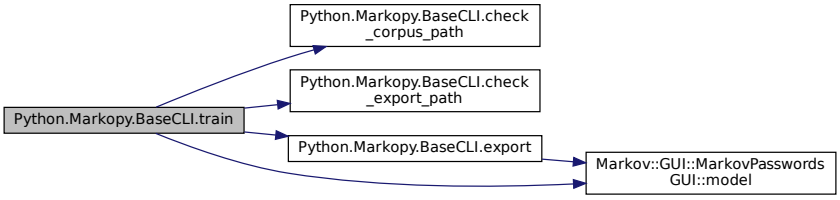
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

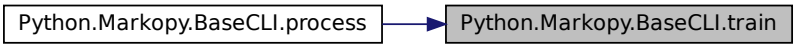
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
       bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
```

```
00104         logging.pprint("Training.")
00105
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else"} and -s/--seperator parameters. Exiting.")
00108             return False
00109
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135
00136         return True
00137
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



8.3.3 Member Data Documentation

8.3.3.1 args

Python.Markopy.BaseCLI.args [inherited]

Definition at line 75 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.generate, Python.Markopy.MarkopyCLI.help(), Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.parse(), Python.CudaMarkopy.CudaMarkopyCLI.parse_fail(), Python.Markopy.BaseCLI.process(), and Python.Markopy.BaseCLI.train().

8.3.3.2 model

`Python.Markopy.BaseCLI.model [inherited]`
Definition at line 40 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.3.3.3 parser

`Python.Markopy.BaseCLI.parser [inherited]`
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.3.3.4 print_help

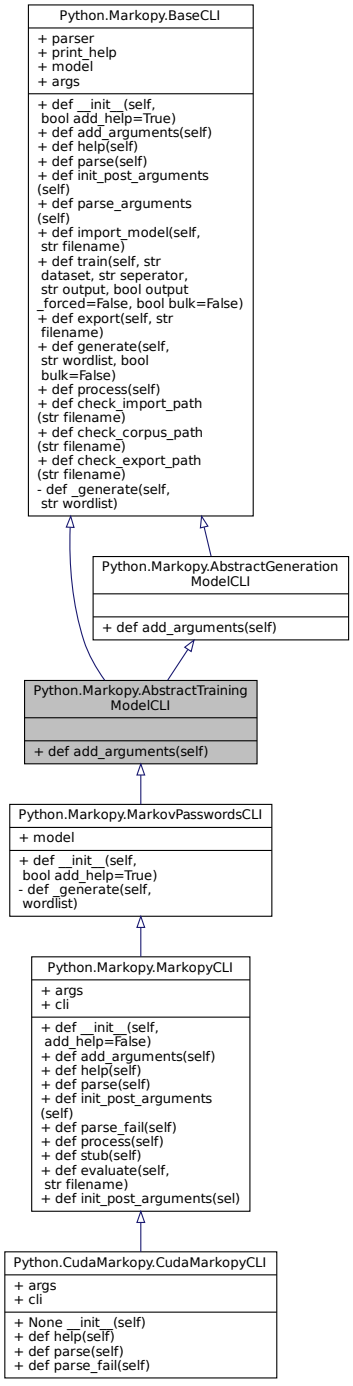
`Python.Markopy.BaseCLI.print_help [inherited]`
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().
The documentation for this class was generated from the following file:

- Markopy/Markopy/src/CLI/base.py

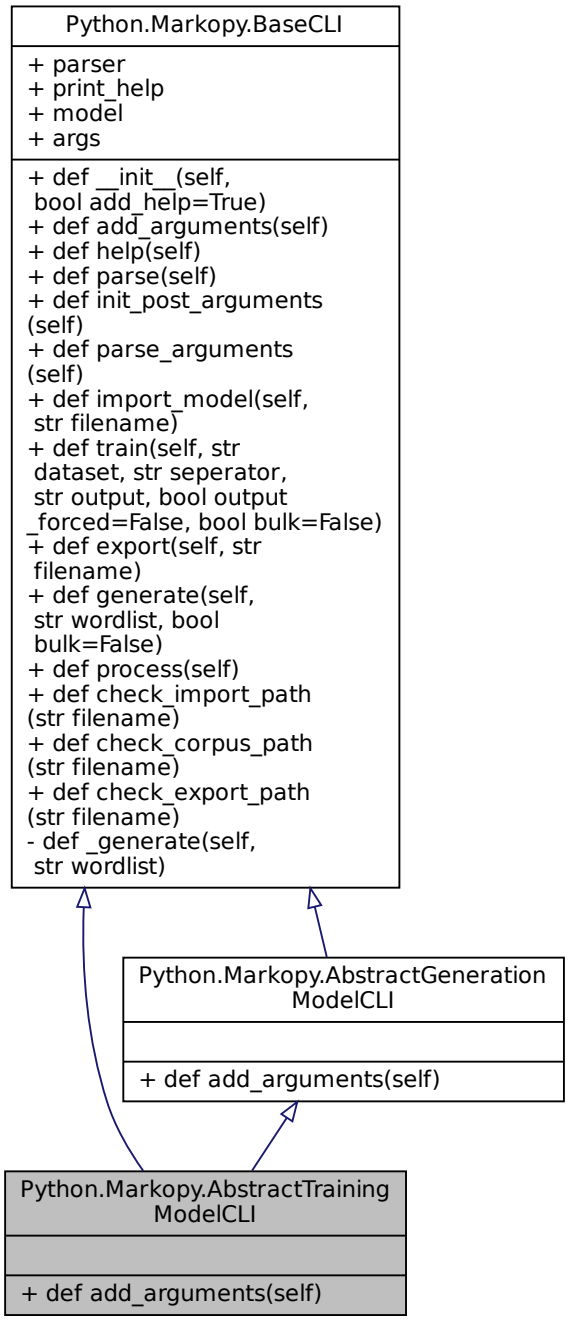
8.4 Python.Markopy.AbstractTrainingModelCLI Class Reference

abstract class for training capable models

Inheritance diagram for Python.Markopy.AbstractTrainingModelCLI:



Collaboration diagram for Python.Markopy.AbstractTrainingModelCLI:



Public Member Functions

- def add_arguments (self)
- def help (self)
- def parse (self)
- def init_post_arguments (self)
- def parse_arguments (self)
- def import_model (self, str filename)

Import a model file.

- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def process (self)

  Process parameters for operation.
- def help (self)
- def parse (self)
- def init_post_arguments (self)
- def parse_arguments (self)
- def import_model (self, str filename)

  Import a model file.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def process (self)

  Process parameters for operation.

Static Public Member Functions

- def check_import_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- def check_import_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity

Public Attributes

- parser
- print_help
- model
- args
- parser
- print_help
- model
- args

Private Member Functions

- def _generate (self, str wordlist)

  wrapper for generate function.

8.4.1 Detailed Description

abstract class for training capable models
Definition at line 274 of file base.py.

8.4.2 Member Function Documentation

8.4.2.1 _generate()

```
def Python.Markopy.BaseCLI._generate (
              self,
             str wordlist )  [private], [inherited]
```

wrapper for generate function.
This can be overloaded by other models

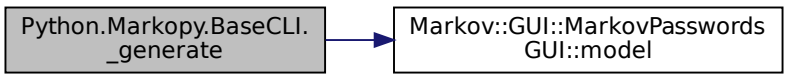
Parameters

wordlist	filename to generate to

Reimplemented in Python.Markopy.ModelMatrixCLI, and Python.CudaMarkopy.CudaModelMatrixCLI.
Definition at line 161 of file base.py.

```
00161     def _generate(self, wordlist : str):
00162         """!
00163         @brief wrapper for generate function. This can be overloaded by other models
00164         @param wordlist filename to generate to
00165         """
00166         self.model.Generate(int(self.args.count), wordlist, int(self.args.min), int(self.args.max),
      int(self.args.threads))
00167
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.generate().
Here is the call graph for this function:



Here is the caller graph for this function:

8.4.2.2 add_arguments()

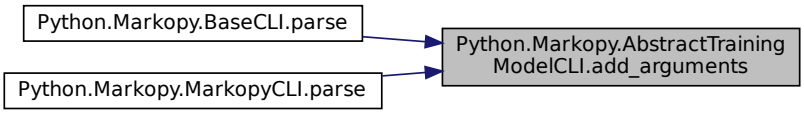
```
def Python.Markopy.AbstractTrainingModelCLI.add_arguments (
              self )
```

Reimplemented from Python.Markopy.BaseCLI.

Reimplemented in Python.Markopy.MarkopyCLI.
Definition at line 282 of file base.py.

```
00282     def add_arguments(self):
00283         "Add command line arguements to the parser"
00284         self.parser.add_argument("-o", "--output",                  help="Output model file. This
      model will be exported when done. Will be ignored for generation mode.")
00285         self.parser.add_argument("-d", "--dataset",                 help="Dataset file to read input
      from for training. Will be ignored for generation mode.")
00286         self.parser.add_argument("-s", "--seperator",               help="Seperator character to use
      with training data.(character between occurrence and value)")
00287         super().add_arguments()
```

References Python.Markopy.BaseCLI.parser.
Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().
Here is the caller graph for this function:



8.4.2.3 check_corpus_path() [1/2]

```
def Python.Markopy.BaseCLI.check_corpus_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186 
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.4.2.4 check_corpus_path() [2/2]

```
def Python.Markopy.BaseCLI.check_corpus_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186 
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.4.2.5 check_export_path() [1/2]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197 
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.4.2.6 check_export_path() [2/2]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197 
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.4.2.7 check_import_path() [1/2]

```
def Python.Markopy.BaseCLI.check_import_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

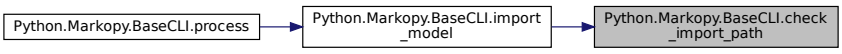
filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
```

Referenced by Python.Markopy.BaseCLI.import_model().

Here is the caller graph for this function:



8.4.2.8 check_import_path() [2/2]

```
def Python.Markopy.BaseCLI.check_import_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

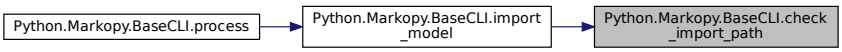
filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
```

Referenced by Python.Markopy.BaseCLI.import_model().

Here is the caller graph for this function:



8.4.2.9 export() [1/2]

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

Parameters

filename	filename to export to

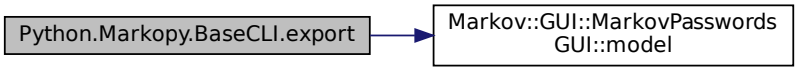
Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144 
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.train().

Here is the call graph for this function:



Here is the caller graph for this function:

8.4.2.10 export() [2/2]

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

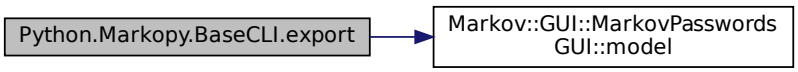
Parameters

filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.4.2.11 generate() [1/2]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

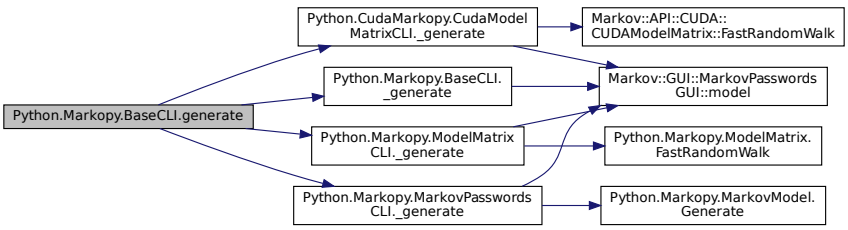
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.4.2.12 generate() [2/2]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename

Parameters

bulk	marks bulk operation with directories

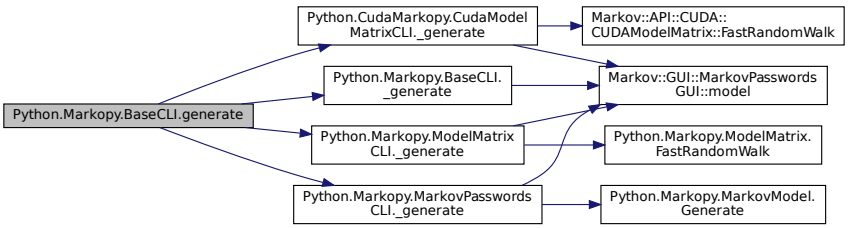
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.4.2.13 help() [1/2]

```
def Python.Markopy.BaseCLI.help (
              self )  [inherited]
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.

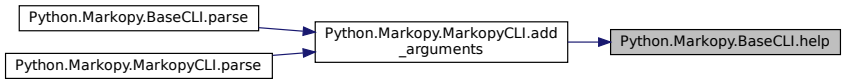
Definition at line 51 of file base.py.

```
00051     def help(self):
00052         "! @brief Handle help strings. Defaults to argparse's help"
00053         self.print_help()
00054
```

References Python.Markopy.BaseCLI.print_help.

Referenced by Python.Markopy.MarkopyCLI.add_arguments().

Here is the caller graph for this function:



8.4.2.14 help() [2/2]

```
def Python.Markopy.BaseCLI.help (
              self )  [inherited]
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.

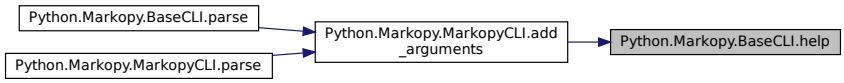
Definition at line 51 of file base.py.

```
00051     def help(self):
00052         "! @brief Handle help strings. Defaults to argparse's help"
00053         self.print_help()
00054
```

References Python.Markopy.BaseCLI.print_help.

Referenced by Python.Markopy.MarkopyCLI.add_arguments().

Here is the caller graph for this function:



8.4.2.15 import_model() [1/2]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

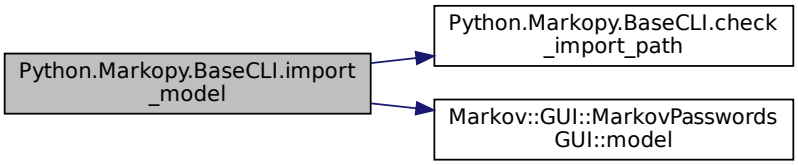
filename	filename to import

Definition at line 77 of file base.py.

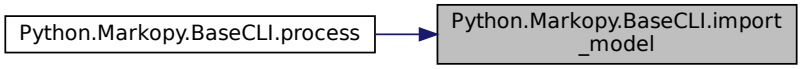
```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model,

Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



8.4.2.16 import_model() [2/2]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

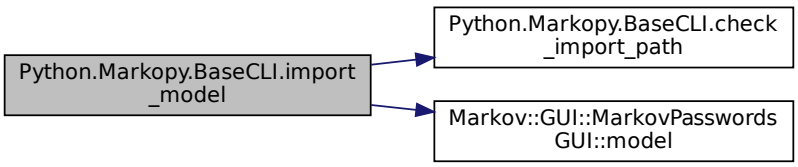
filename	filename to import

Definition at line 77 of file base.py.

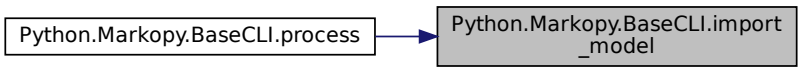
```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.2.17 init_post_arguments() [1/2]

```
def Python.Markopy.BaseCLI.init_post_arguments (
              self )  [inherited]
```

Reimplemented in Python.Markopy.ModelMatrixCLI, Python.Markopy.MarkopyCLI, Python.CudaMarkopy.CudaModelMatrixCLI, and Python.Markopy.MarkopyCLI.

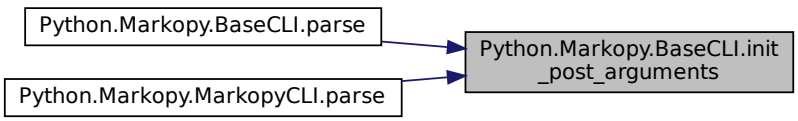
Definition at line 62 of file base.py.

```
00062     def init_post_arguments(self):
00063         "! @brief set up stuff that is collected from command line arguements"
00064         logging.VERBOSITY = 0
00065         try:
00066             if self.args.verbosity:
00067                 logging.VERBOSITY = self.args.verbosity
00068                 logging.pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00069         except:
00070             pass
00071
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



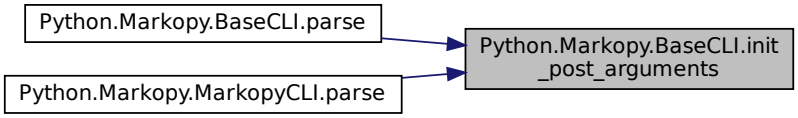
8.4.2.18 init_post_arguments() [2/2]

```
def Python.Markopy.BaseCLI.init_post_arguments (
              self )  [inherited]
```

Reimplemented in Python.Markopy.ModelMatrixCLI, Python.Markopy.MarkopyCLI, Python.CudaMarkopy.CudaModelMatrixCLI, and Python.Markopy.MarkopyCLI.
Definition at line 62 of file base.py.

```
00062     def init_post_arguments(self):
00063         "! @brief set up stuff that is collected from command line arguements"
00064         logging.VERBOSITY = 0
00065         try:
00066             if self.args.verbosity:
00067                 logging.VERBOSITY = self.args.verbosity
00068                 logging.pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00069         except:
00070             pass
00071
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.
Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().
Here is the caller graph for this function:



8.4.2.19 parse() [1/2]

```
def Python.Markopy.BaseCLI.parse (
              self )  [inherited]
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.
Definition at line 55 of file base.py.

```
00055     def parse(self):
00056         "! @brief add, parse and hook arguements"
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMode
Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.init_post_arguments(), Python.Markopy.ModelMatrixC
and Python.Markopy.BaseCLI.parse_arguments().

Here is the call graph for this function:

8.4.2.20 parse() [2/2]

```
def Python.Markopy.BaseCLI.parse (
              self )  [inherited]
```

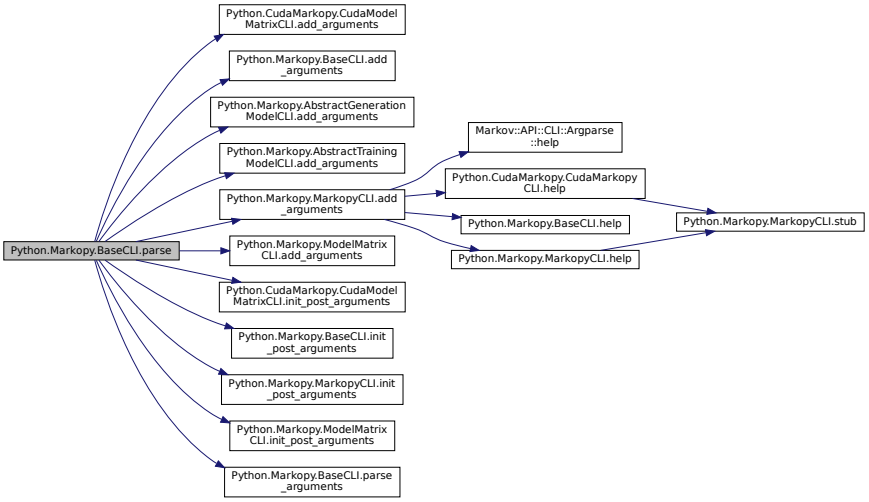
Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.

Definition at line 55 of file base.py.

```
00055     def parse(self):
00056         "! @brief add, parse and hook arguements"
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMode Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.init_post_arguments(), Python.Markopy.ModelMatrixC and Python.Markopy.BaseCLI.parse_arguments().

Here is the call graph for this function:



8.4.2.21 parse_arguments() [1/2]

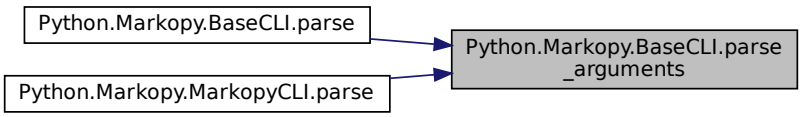
```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.4.2.22 parse_arguments() [2/2]

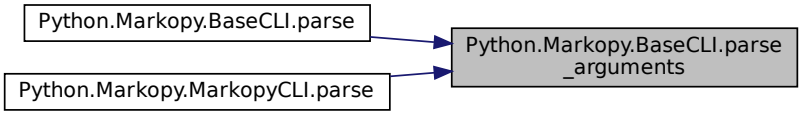
```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.4.2.23 process() [1/2]

```
def Python.Markopy.BaseCLI.process (
            self )  [inherited]
```

Process parameters for operation.

Reimplemented in Python.Markopy.MarkopyCLI.

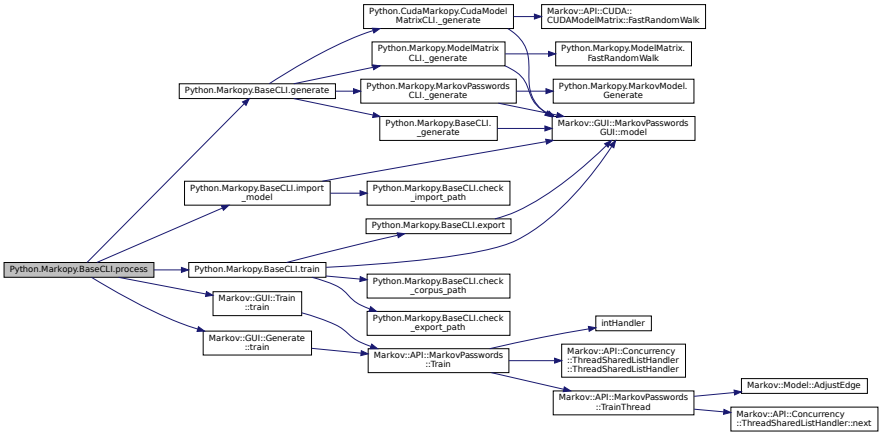
Definition at line 202 of file base.py.

```
00202     def process(self):
00203         """!
00204         @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging.pprint(f"Bulk mode operation chosen.", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
      (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00210                     corpus_list = os.listdir(self.args.dataset)
00211                     for corpus in corpus_list:
00212                         self.import_model(self.args.input)
00213                         logging.pprint(f"Training {self.args.input} with {corpus}", 2)
00214                         output_file_name = corpus
00215                         model_extension = ""
00216                         if "." in self.args.input:
00217                             model_extension = self.args.input.split(".")[-1]
00218                         self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
      f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00219                 else:
00220                     logging.pprint("In bulk training, output and dataset should be a directory.")
00221                     exit(1)
00222
00223             elif (self.args.mode.lower() == "generate"):
00224                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
      (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00225                     model_list = os.listdir(self.args.input)
00226                     print(model_list)
00227                     for input in model_list:
00228                         logging.pprint(f"Generating from {self.args.input}/{input} to
      {self.args.wordlist}/{input}.txt", 2)
00229                         self.import_model(f"{self.args.input}/{input}")
00230                         model_base = input
00231                         if "." in self.args.input:
00232                             model_base = input.split(".")[1]
00233                         self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00234                 else:
00235                     logging.pprint("In bulk generation, input and wordlist should be directory.")
00236
00237         else:
00238             self.import_model(self.args.input)
00239             if (self.args.mode.lower() == "generate"):
00240                 self.generate(self.args.wordlist)
00241
00242
00243             elif (self.args.mode.lower() == "train"):
00244                 self.train(self.args.dataset, self.args.seperator, self.args.output,
      output_forced=True)
00245
00246
00247             elif(self.args.mode.lower() == "combine"):
00248                 self.train(self.args.dataset, self.args.seperator, self.args.output)
00249                 self.generate(self.args.wordlist)
00250
00251
00252             else:
00253                 logging.pprint("Invalid mode arguement given.")
```

```
00254                 logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255                 exit(5)
00256
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.generate(), Python.Markopy.BaseCLI.import_model(), Markov::GUI::Generate.train(), Markov::GUI::Train.train(), and Python.Markopy.BaseCLI.train().

Here is the call graph for this function:



8.4.2.24 process() [2/2]

```
def Python.Markopy.BaseCLI.process (
            self ) [inherited]
```

Process parameters for operation.

Reimplemented in Python.Markopy.MarkopyCLI.

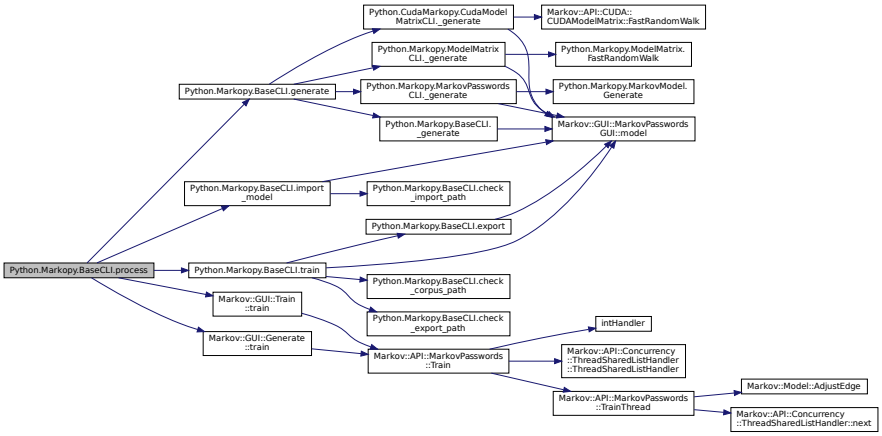
Definition at line 202 of file base.py.

```
00202     def process(self):
00203         """!
00204         @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging.pprint(f"Bulk mode operation chosen.", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
      (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00210                     corpus_list = os.listdir(self.args.dataset)
00211                     for corpus in corpus_list:
00212                         self.import_model(self.args.input)
00213                         logging.pprint(f"Training {self.args.input} with {corpus}", 2)
00214                         output_file_name = corpus
00215                         model_extension = ""
00216                         if "." in self.args.input:
00217                             model_extension = self.args.input.split(".")[-1]
00218                         self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
      f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00219                 else:
00220                     logging.pprint("In bulk training, output and dataset should be a directory.")
00221                     exit(1)
00222
00223             elif (self.args.mode.lower() == "generate"):
00224                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
      (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00225                     model_list = os.listdir(self.args.input)
00226                     print(model_list)
00227                     for input in model_list:
00228                         logging.pprint(f"Generating from {self.args.input}/{input} to
      {self.args.wordlist}/{input}.txt", 2)
00229                         self.import_model(f"{self.args.input}/{input}")
00230                         model_base = input
00231                         if "." in self.args.input:
00232                             model_base = input.split(".")[1]
00233                         self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00234                 else:
00235                     logging.pprint("In bulk generation, input and wordlist should be directory.")
```

```
00236
00237         else:
00238             self.import_model(self.args.input)
00239             if (self.args.mode.lower() == "generate"):
00240                 self.generate(self.args.wordlist)
00241
00242
00243             elif (self.args.mode.lower() == "train"):
00244                 self.train(self.args.dataset, self.args.seperator, self.args.output,
      output_forced=True)
00245
00246
00247             elif(self.args.mode.lower() == "combine"):
00248                 self.train(self.args.dataset, self.args.seperator, self.args.output)
00249                 self.generate(self.args.wordlist)
00250
00251
00252             else:
00253                 logging.pprint("Invalid mode arguement given.")
00254                 logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255                 exit(5)
00256
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.generate(), Python.Markopy.BaseCLI.import_model(), Markov::GUI::Generate.train(), Markov::GUI::Train.train(), and Python.Markopy.BaseCLI.train().

Here is the call graph for this function:



8.4.2.25 train() [1/2]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

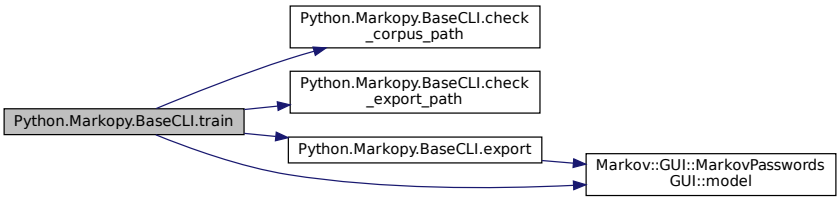
Definition at line 94 of file base.py.

```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else''} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

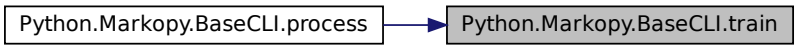
References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.2.26 train() [2/2]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

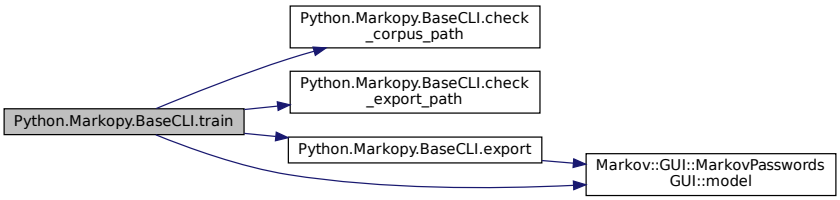
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

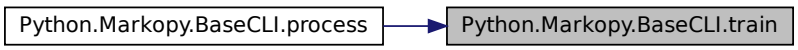
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
```

```
00128         logging.pprint(f'Training completed.', 2)
00129
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135
00136         return True
00137
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3 Member Data Documentation

8.4.3.1 args [1/2]

Python.Markopy.BaseCLI.args [inherited]
Definition at line 75 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.generate(), Python.Markopy.MarkopyCLI.help(), Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.parse(), Python.CudaMarkopy.CudaMarkopyCLI.parse_fail(), Python.Markopy.BaseCLI.process(), and Python.Markopy.BaseCLI.train().

8.4.3.2 args [2/2]

Python.Markopy.BaseCLI.args [inherited]
Definition at line 75 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.generate(), Python.Markopy.MarkopyCLI.help(), Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.parse(), Python.CudaMarkopy.CudaMarkopyCLI.parse_fail(), Python.Markopy.BaseCLI.process(), and Python.Markopy.BaseCLI.train().

8.4.3.3 model [1/2]

Python.Markopy.BaseCLI.model [inherited]
Definition at line 40 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.4.3.4 model [2/2]

Python.Markopy.BaseCLI.model [inherited]
Definition at line 40 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.4.3.5 parser [1/2]

Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.4.3.6 parser [2/2]

Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.4.3.7 print_help [1/2]

Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.4.3.8 print_help [2/2]

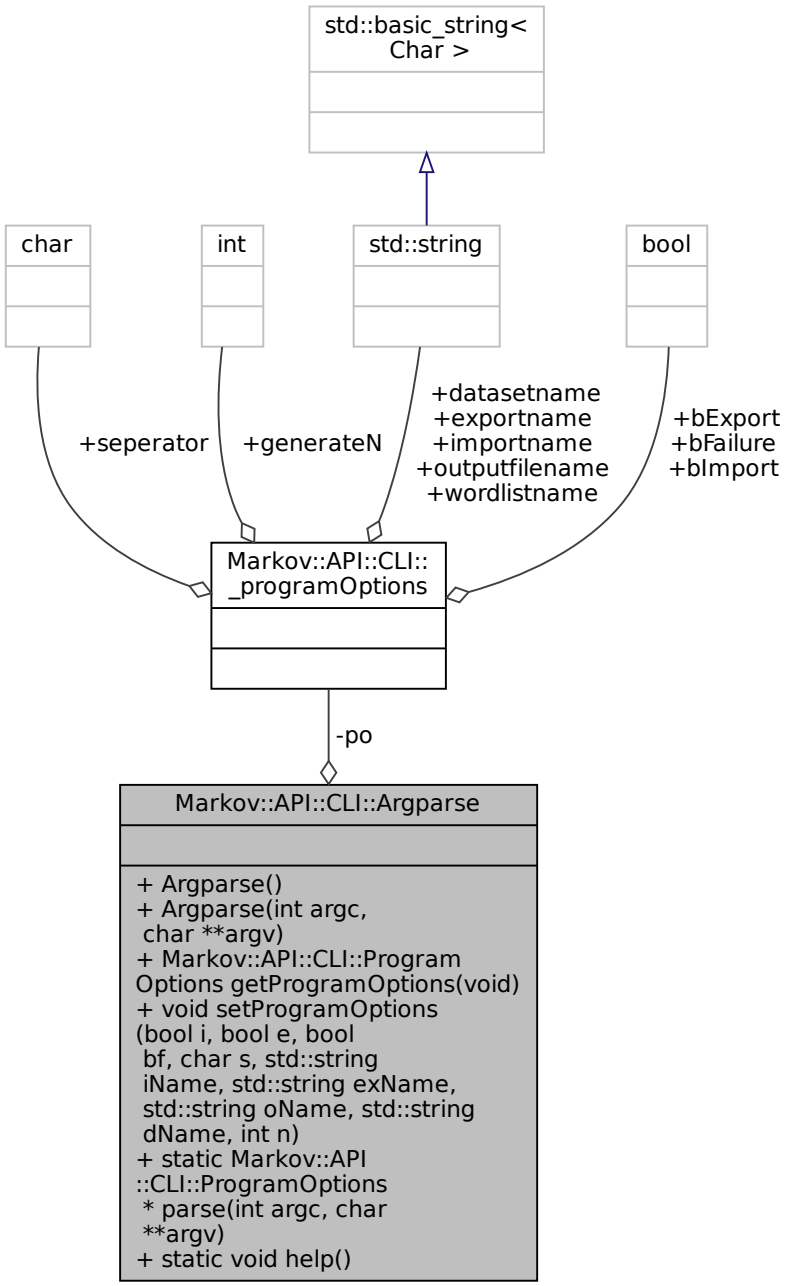
Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().
The documentation for this class was generated from the following file:

- Markopy/Markopy/src/CLI/base.py

8.5 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

  ProgramOptions structure object.

8.5.1 Detailed Description

Parse command line arguements.
Definition at line 82 of file argparse.h.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 Argparse() [1/2]

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

8.5.2.2 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

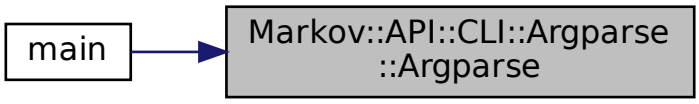
Definition at line 94 of file argparse.h.

```
00094                                       {
00095
00096             /*bool bImp;
00097             bool bExp;
00098             bool bFail;
00099             char sprt;
00100             std::string imports;
00101             std::string exports;
00102             std::string outputs;
00103             std::string datasets;
00104             int generateN;
00105             */
00106             opt::options_description desc("Options");
00107
00108
00109             desc.add_options()
00110                 ("generate", "Generate strings with given parameters")
00111                 ("train", "Train model with given parameters")
00112                 ("combine", "Combine")
00113                 ("import", opt::value<std::string>(), "Import model file")
00114                 ("output", opt::value<std::string>(), "Output model file. This model will be exported
       when done. Will be ignored for generation mode")
```

```
00115                 ("dataset", opt::value<std::string>(), "Dataset file to read input from training. Will
      be ignored for generation mode")
00116                 ("seperator", opt::value<char>(), "Seperator character to use with training data.
      (character between occurence and value)")
00117                 ("wordlist", opt::value<std::string>(), "Wordlist file path to export generation
      results to. Will be ignored for training mode")
00118                 ("count", opt::value<int>(), "Number of lines to generate. Ignored in training mode")
00119                 ("verbosity", "Output verbosity")
00120                 ("help", "Option definitions");
00121
00122                 opt::variables_map vm;
00123
00124                 opt::store(opt::parse_command_line(argc, argv, desc), vm);
00125
00126                 opt::notify(vm);
00127
00128                 //std::cout << desc << std::endl;
00129                 if (vm.count("help")) {
00130                 std::cout << desc << std::endl;
00131                 }
00132
00133                 if (vm.count("output") == 0) this->po.outputfilename = "NULL";
00134                 else if (vm.count("output") == 1) {
00135                     this->po.outputfilename = vm["output"].as<std::string>();
00136                     this->po.bExport = true;
00137                 }
00138                 else {
00139                     this->po.bFailure = true;
00140                     std::cout << "UNIDENTIFIED INPUT" << std::endl;
00141                     std::cout << desc << std::endl;
00142                 }
00143
00144
00145                 if (vm.count("dataset") == 0) this->po.datasetname = "NULL";
00146                 else if (vm.count("dataset") == 1) {
00147                     this->po.datasetname = vm["dataset"].as<std::string>();
00148                 }
00149                 else {
00150                     this->po.bFailure = true;
00151                     std::cout << "UNIDENTIFIED INPUT" << std::endl;
00152                     std::cout << desc << std::endl;
00153                 }
00154
00155
00156                 if (vm.count("wordlist") == 0) this->po.wordlistname = "NULL";
00157                 else if (vm.count("wordlist") == 1) {
00158                     this->po.wordlistname = vm["wordlist"].as<std::string>();
00159                 }
00160                 else {
00161                     this->po.bFailure = true;
00162                     std::cout << "UNIDENTIFIED INPUT" << std::endl;
00163                     std::cout << desc << std::endl;
00164                 }
00165
00166                 if (vm.count("import") == 0) this->po.importname = "NULL";
00167                 else if (vm.count("import") == 1) {
00168                     this->po.importname = vm["import"].as<std::string>();
00169                     this->po.bImport = true;
00170                 }
00171                 else {
00172                     this->po.bFailure = true;
00173                     std::cout << "UNIDENTIFIED INPUT" << std::endl;
00174                     std::cout << desc << std::endl;
00175                 }
00176
00177
00178                 if (vm.count("count") == 0) this->po.generateN = 0;
00179                 else if (vm.count("count") == 1) {
00180                     this->po.generateN = vm["count"].as<int>();
00181                 }
00182                 else {
00183                     this->po.bFailure = true;
00184                     std::cout << "UNIDENTIFIED INPUT" << std::endl;
00185                     std::cout << desc << std::endl;
00186                 }
00187
00188                 /*std::cout << vm["output"].as<std::string>() << std::endl;
00189                 std::cout << vm["dataset"].as<std::string>() << std::endl;
00190                 std::cout << vm["wordlist"].as<std::string>() << std::endl;
00191                 std::cout << vm["output"].as<std::string>() << std::endl;
00192                 std::cout << vm["count"].as<int>() << std::endl;*/
00193
00194
00195                 //else if (vm.count("train")) std::cout << "train oldu" << std::endl;
00196         }
```

References Markov::API::CLI::_programOptions::bExport, Markov::API::CLI::_programOptions::bFailure, Markov::API::CLI::_program

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



8.5.3 Member Function Documentation

8.5.3.1 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 203 of file argparse.h.

```
00203                                                   {
00204             return this->po;
00205         }
```

References po.

8.5.3.2 help()

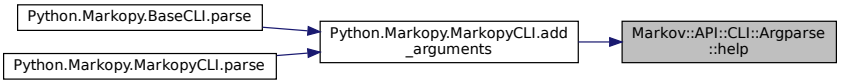
```
void Markov::API::CLI::Argparse::help ( )  [static]
```

Print help string.
Definition at line 15 of file argparse.cpp.

```
00015                                {
00016     std::cout «
00017     "Markov Passwords - Help\n"
00018     "Options:\n"
00019     "   \n"
00020     "   -of --outputfilename\n"
00021     "       Filename to output the generation results\n"
00022     "   -ef --exportfilename\n"
00023     "       filename to export built model to\n"
00024     "   -if --importfilename\n"
00025     "       filename to import model from\n"
00026     "   -n (generate count)\n"
00027     "       Number of lines to generate\n"
00028     "   \n"
00029     "Usage: \n"
00030     "   markov.exe -if empty_model.mdl -ef model.mdl\n"
00031     "       import empty_model.mdl and train it with data from stdin. When done, output the model to
      model.mdl\n"
00032     "\n"
00033     "   markov.exe -if empty_model.mdl -n 15000 -of wordlist.txt\n"
00034     "       import empty_model.mdl and generate 15000 words to wordlist.txt\n"
00035
00036      « std::endl;
00037 }
```

Referenced by Python.Markopy.MarkopyCLI::add_arguments().

Here is the caller graph for this function:



8.5.3.3 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 11 of file argparse.cpp.

```
00011 { return 0; }
```

8.5.3.4 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 220 of file argparse.h.

```
00220                                                {
00221             this->po.bImport = i;
00222             this->po.bExport = e;
00223             this->po.seperator = s;
00224             this->po.bFailure = bf;
00225             this->po.generateN = n;
00226             this->po.importname = iName;
00227             this->po.exportname = exName;
00228             this->po.outputfilename = oName;
00229             this->po.datasetname = dName;
00230 
00231             /*strcpy_s(this->po.importname,256,iName);
00232             strcpy_s(this->po.exportname,256,exName);
00233             strcpy_s(this->po.outputfilename,256,oName);
00234             strcpy_s(this->po.datasetname,256,dName);*/
00235 
00236         }
```

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

8.5.4 Member Data Documentation

8.5.4.1 po

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

ProgramOptions structure object.

Definition at line 255 of file argparse.h.

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

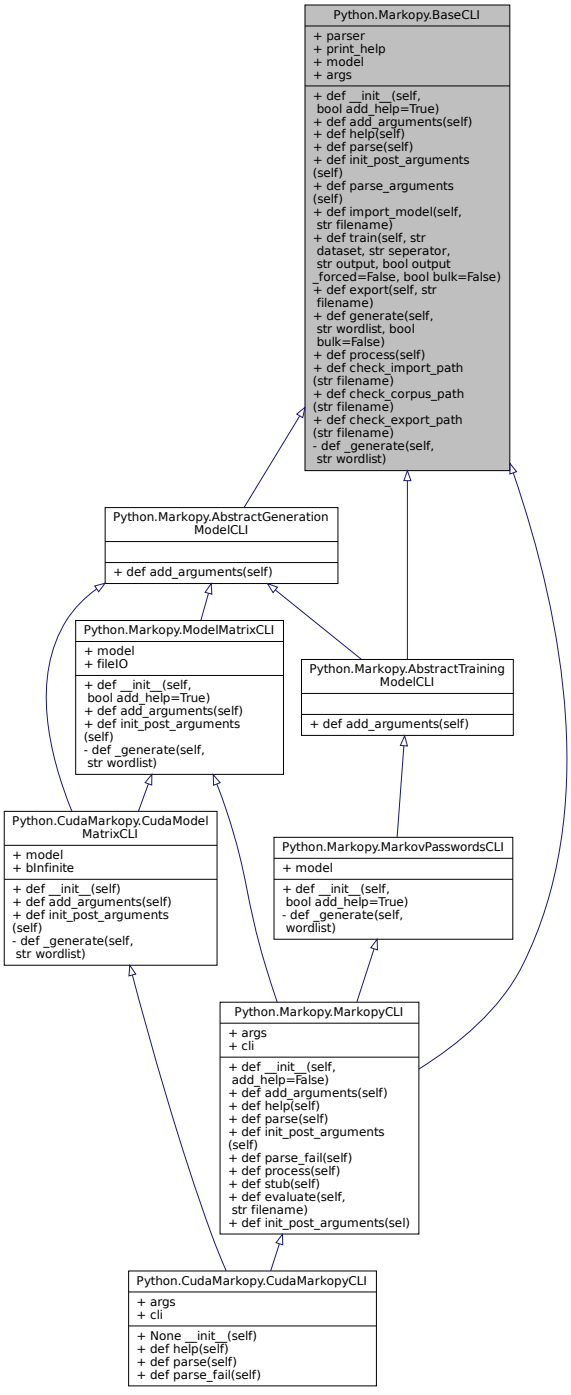
The documentation for this class was generated from the following files:

- Markopy/MarkovAPICLI/src/argparse.h
- Markopy/MarkovAPICLI/src/argparse.cpp

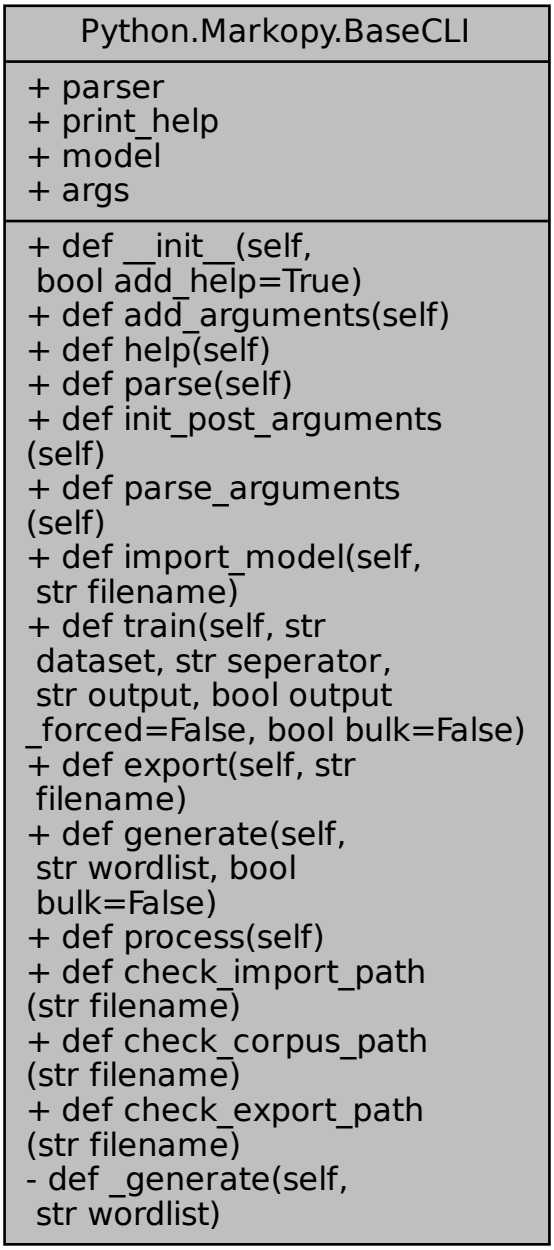
8.6 Python.Markopy.BaseCLI Class Reference

Base CLI class to handle user interactions

Inheritance diagram for Python.Markopy.BaseCLI:



Collaboration diagram for Python.Markopy.BaseCLI:



Public Member Functions

- def __init__ (self, bool add_help=True)

  initialize base CLI
- def add_arguments (self)
- def help (self)
- def parse (self)
- def init_post_arguments (self)

- def parse_arguments (self)
- def import_model (self, str filename)

  Import a model file.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def process (self)

  Process parameters for operation.

Static Public Member Functions

- def check_import_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity

Public Attributes

- parser
- print_help
- model
- args

Private Member Functions

- def _generate (self, str wordlist)

  wrapper for generate function.

8.6.1 Detailed Description

Base CLI class to handle user interactions

Definition at line 16 of file base.py.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 __init__()

```
def Python.Markopy.BaseCLI.__init__ (
            self,
          bool add_help = True )
```

initialize base CLI

Parameters

add_help	decide to overload the help function or not

Reimplemented in Python.Markopy.MarkovPasswordsCLI, and Python.Markopy.ModelMatrixCLI.

Definition at line 20 of file base.py.

```
00020     def __init__(self, add_help : bool=True):
00021         """!
00022         @brief initialize base CLI
00023         @param add_help decide to overload the help function or not
00024         """
00025         self.parser = argparse.ArgumentParser(description="Python wrapper for MarkovPasswords.",
00026         epilog=f"""{colored("Sample runs:", "yellow")}
00027         {__file__.split("/")[-1]} train untrained.mdl -d dataset.dat -s "\\t" -o trained.mdl
00028             Import untrained.mdl, train it with dataset.dat which has tab delimited data, output
      resulting model to trained.mdl\n
00029
00030         {__file__.split("/")[-1]} generate trained.mdl -n 500 -w output.txt
00031             Import trained.mdl, and generate 500 lines to output.txt
00032
00033         {__file__.split("/")[-1]} combine untrained.mdl -d dataset.dat -s "\\t" -n 500 -w output.txt
00034             Train and immediately generate 500 lines to output.txt. Do not export trained model.
00035
00036         {__file__.split("/")[-1]} combine untrained.mdl -d dataset.dat -s "\\t" -n 500 -w output.txt
      -o trained.mdl
00037             Train and immediately generate 500 lines to output.txt. Export trained model.
00038         """, add_help=add_help, formatter_class=argparse.RawTextHelpFormatter)
00039         self.print_help = self.parser.print_help
00040         self.model = MarkovModel()
00041
```

8.6.3 Member Function Documentation

8.6.3.1 _generate()

```
def Python.Markopy.BaseCLI._generate (
              self,
             str wordlist )  [private]
```

wrapper for generate function.
This can be overloaded by other models

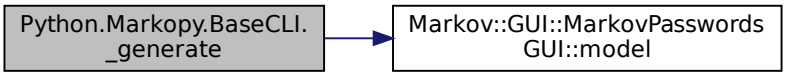
Parameters

wordlist	filename to generate to

Reimplemented in Python.Markopy.ModelMatrixCLI, and Python.CudaMarkopy.CudaModelMatrixCLI.
Definition at line 161 of file base.py.

```
00161     def _generate(self, wordlist : str):
00162         """!
00163         @brief wrapper for generate function. This can be overloaded by other models
00164         @param wordlist filename to generate to
00165         """
00166         self.model.Generate(int(self.args.count), wordlist, int(self.args.min), int(self.args.max),
      int(self.args.threads))
00167
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.generate().
Here is the call graph for this function:



Here is the caller graph for this function:

8.6.3.2 add_arguments()

```
def Python.Markopy.BaseCLI.add_arguments (
              self )
```

Reimplemented in Python.Markopy.ModelMatrixCLI, Python.Markopy.MarkopyCLI, Python.Markopy.AbstractTrainingModelCLI, Python.Markopy.AbstractGenerationModelCLI, and Python.CudaMarkopy.CudaModelMatrixCLI.

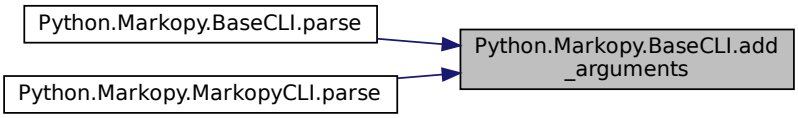
Definition at line 43 of file base.py.

```
00043     def add_arguments(self):
00044         "! @brief Add command line arguements to the parser"
00045         self.parser.add_argument("mode",                              help="Process mode. Either
      'Train', 'Generate', or 'Combine'.")
00046         self.parser.add_argument("-t", "--threads",default=10,        help="Number of lines to
      generate. Ignored in training mode.")
00047         self.parser.add_argument("-v", "--verbosity",action="count", help="Output verbosity.")
00048         self.parser.add_argument("-b", "--bulk",action="store_true", help="Bulk generate or bulk train
      every corpus/model in the folder.")
00049
```

References Python.Markopy.BaseCLI.parser.

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.6.3.3 check_corpus_path()

```
def Python.Markopy.BaseCLI.check_corpus_path (
             str filename )  [static]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.6.3.4 check_export_path()

```
def Python.Markopy.BaseCLI.check_export_path (
            str filename )  [static]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197 
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.6.3.5 check_import_path()

```
def Python.Markopy.BaseCLI.check_import_path (
            str filename )  [static]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:

8.6.3.6 export()

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )
```

Export model to a file.

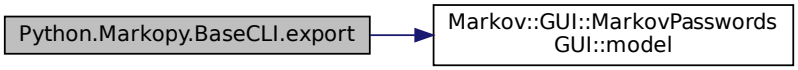
Parameters

filename	filename to export to

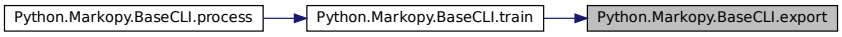
Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.6.3.7 generate()

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

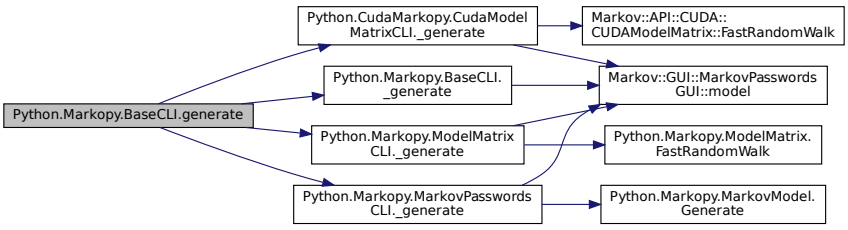
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.6.3.8 help()

```
def Python.Markopy.BaseCLI.help (
            self )
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.

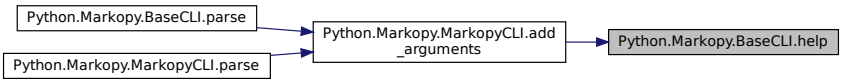
Definition at line 51 of file base.py.

```
00051     def help(self):
00052         "! @brief Handle help strings. Defaults to argparse's help"
00053         self.print_help()
00054
```

References Python.Markopy.BaseCLI.print_help.

Referenced by Python.Markopy.MarkopyCLI.add_arguments().

Here is the caller graph for this function:



8.6.3.9 import_model()

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )
```

Import a model file.

Parameters

filename	filename to import

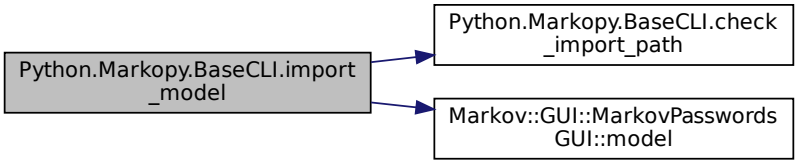
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

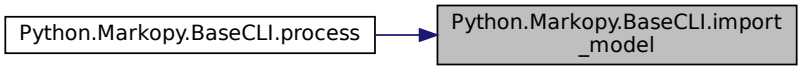
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.6.3.10 init_post_arguments()

```
def Python.Markopy.BaseCLI.init_post_arguments (
              self )
```

Reimplemented in Python.Markopy.ModelMatrixCLI, Python.Markopy.MarkopyCLI, Python.CudaMarkopy.CudaModelMatrixCLI, and Python.Markopy.MarkopyCLI.

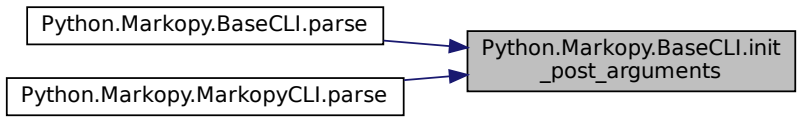
Definition at line 62 of file base.py.

```
00062     def init_post_arguments(self):
00063         "! @brief set up stuff that is collected from command line arguements"
00064         logging.VERBOSITY = 0
00065         try:
00066             if self.args.verbosity:
00067                 logging.VERBOSITY = self.args.verbosity
00068                 logging.pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00069         except:
00070             pass
00071
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.6.3.11 parse()

```
def Python.Markopy.BaseCLI.parse (
              self )
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.

Definition at line 55 of file base.py.

```
00055     def parse(self):
00056         "! @brief add, parse and hook arguements"
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMode
Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.init_post_arguments(), Python.Markopy.ModelMatrixC
and Python.Markopy.BaseCLI.parse_arguments().

Here is the call graph for this function:

8.6.3.12 parse_arguments()

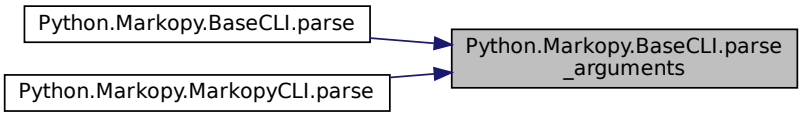
```
def Python.Markopy.BaseCLI.parse_arguments (
              self )
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.6.3.13 process()

```
def Python.Markopy.BaseCLI.process (
              self )
```

Process parameters for operation.

Reimplemented in Python.Markopy.MarkopyCLI.

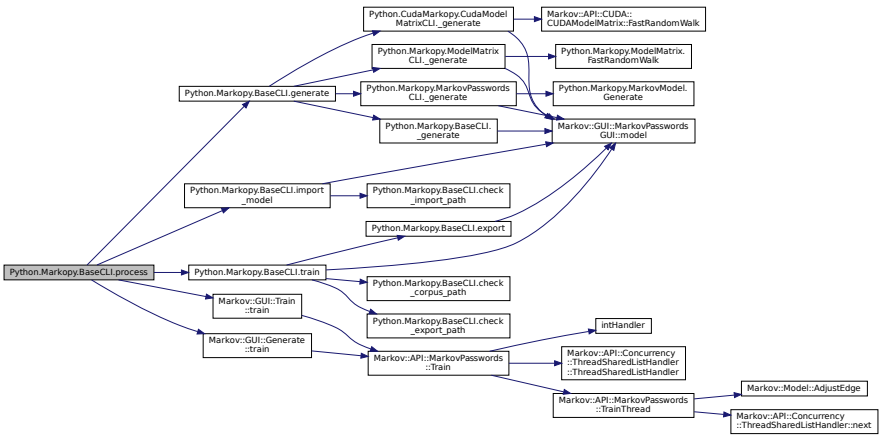
Definition at line 202 of file base.py.

```
00202     def process(self):
00203         """!
00204         @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging.pprint(f"Bulk mode operation chosen.", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
      (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
```

```
00210                   corpus_list = os.listdir(self.args.dataset)
00211                   for corpus in corpus_list:
00212                       self.import_model(self.args.input)
00213                       logging.pprint(f"Training {self.args.input} with {corpus}", 2)
00214                       output_file_name = corpus
00215                       model_extension = ""
00216                       if "." in self.args.input:
00217                           model_extension = self.args.input.split(".")[-1]
00218                       self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
    f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00219               else:
00220                   logging.pprint("In bulk training, output and dataset should be a directory.")
00221                   exit(1)
00222
00223           elif (self.args.mode.lower() == "generate"):
00224               if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
    (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00225                   model_list = os.listdir(self.args.input)
00226                   print(model_list)
00227                   for input in model_list:
00228                       logging.pprint(f"Generating from {self.args.input}/{input} to
    {self.args.wordlist}/{input}.txt", 2)
00229                       self.import_model(f"{self.args.input}/{input}")
00230                       model_base = input
00231                       if "." in self.args.input:
00232                           model_base = input.split(".")[1]
00233                       self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00234               else:
00235                   logging.pprint("In bulk generation, input and wordlist should be directory.")
00236
00237       else:
00238           self.import_model(self.args.input)
00239           if (self.args.mode.lower() == "generate"):
00240               self.generate(self.args.wordlist)
00241
00242
00243           elif (self.args.mode.lower() == "train"):
00244               self.train(self.args.dataset, self.args.seperator, self.args.output,
    output_forced=True)
00245
00246
00247           elif(self.args.mode.lower() == "combine"):
00248               self.train(self.args.dataset, self.args.seperator, self.args.output)
00249               self.generate(self.args.wordlist)
00250
00251
00252           else:
00253               logging.pprint("Invalid mode arguement given.")
00254               logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255               exit(5)
00256
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.generate(), Python.Markopy.BaseCLI.import_model(), Markov::GUI::Generate.train(), Markov::GUI::Train.train(), and Python.Markopy.BaseCLI.train().

Here is the call graph for this function:



8.6.3.14 train()

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )
```

Train a model via CLI parameters.

Parameters

model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

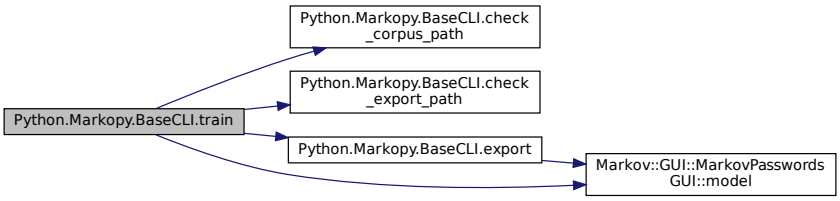
Definition at line 94 of file base.py.

```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
       bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
       else''} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
       accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

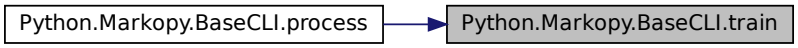
References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.6.4 Member Data Documentation

8.6.4.1 args

`Python.Markopy.BaseCLI.args`
Definition at line 75 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.generate Python.Markopy.MarkopyCLI.help(), Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.parse(), Python.CudaMarkopy.CudaMarkopyCLI.parse_fail(), Python.Markopy.BaseCLI.process(), and Python.Markopy.BaseCLI.train().

8.6.4.2 model

`Python.Markopy.BaseCLI.model`
Definition at line 40 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.6.4.3 parser

`Python.Markopy.BaseCLI.parser`
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.6.4.4 print_help

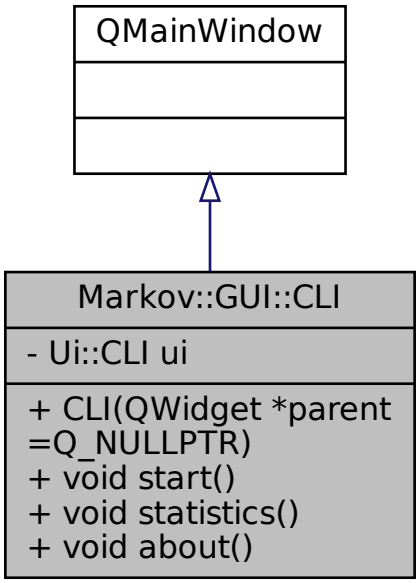
`Python.Markopy.BaseCLI.print_help`

Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().
The documentation for this class was generated from the following file:

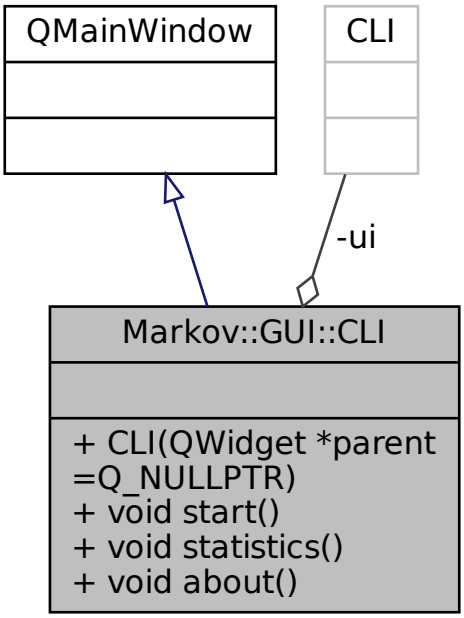
- Markopy/Markopy/src/CLI/base.py

8.7 Markov::GUI::CLI Class Reference

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



Collaboration diagram for Markov::GUI::CLI:



Public Slots

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

Public Member Functions

- CLI (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::CLI ui

8.7.1 Detailed Description

QT CLI Class.
Definition at line 14 of file CLI.h.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 CLI()

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

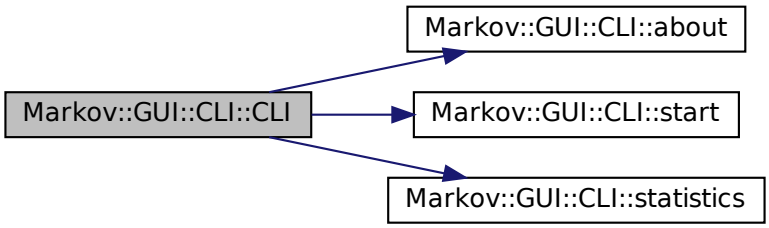
Definition at line 15 of file CLI.cpp.

```
00016     : QMainWindow(parent)
```

```
00017 {
00018     ui.setupUi(this);
00019
00020     QObject::connect(ui.startButton, &QPushButton::clicked, this, [this] {start(); });
00021     QObject::connect(ui.commandLinkButton_2, &QPushButton::clicked, this, [this] {statistics(); });
00022     QObject::connect(ui.commandLinkButton, &QPushButton::clicked, this, [this] {about(); });
00023
00024 }
```

References about(), start(), statistics(), and ui.

Here is the call graph for this function:



8.7.3 Member Function Documentation

8.7.3.1 about

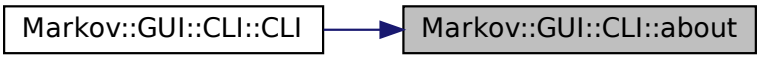
```
void Markov::GUI::CLI::about ( )  [slot]
```

Definition at line 36 of file CLI.cpp.

```
00036                          {
00037     /*
00038     about button
00039     */
00040 }
```

Referenced by CLI().

Here is the caller graph for this function:



8.7.3.2 start

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

Definition at line 26 of file CLI.cpp.

```
00026                          {
00027     Train* w = new Train;
00028     w->show();
00029     this->close();
00030 }
```

Referenced by CLI().

Here is the caller graph for this function:

8.7.3.3 statistics

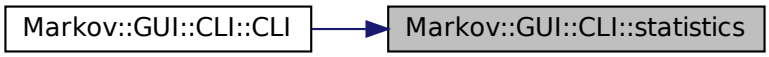
```
void Markov::GUI::CLI::statistics ( )  [slot]
```

Definition at line 31 of file CLI.cpp.

```
00031                                {
00032     /*
00033     statistic will show
00034     */
00035 }
```

Referenced by CLI().

Here is the caller graph for this function:



8.7.4 Member Data Documentation

8.7.4.1 ui

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

Definition at line 20 of file CLI.h.

Referenced by CLI().

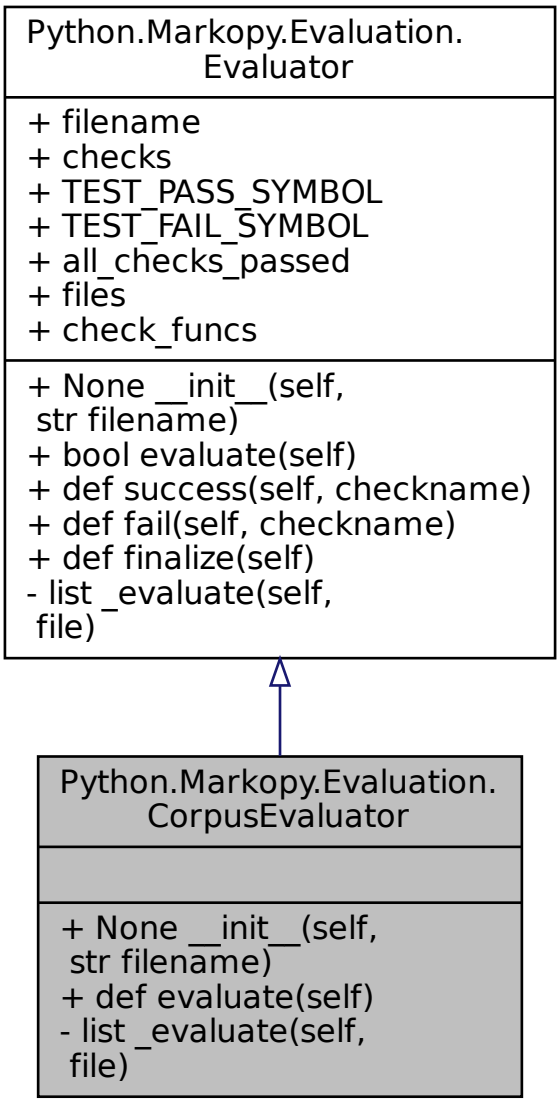
The documentation for this class was generated from the following files:

- Markopy/MarkovPasswordsGUI/src/CLI.h
- Markopy/MarkovPasswordsGUI/src/CLI.cpp

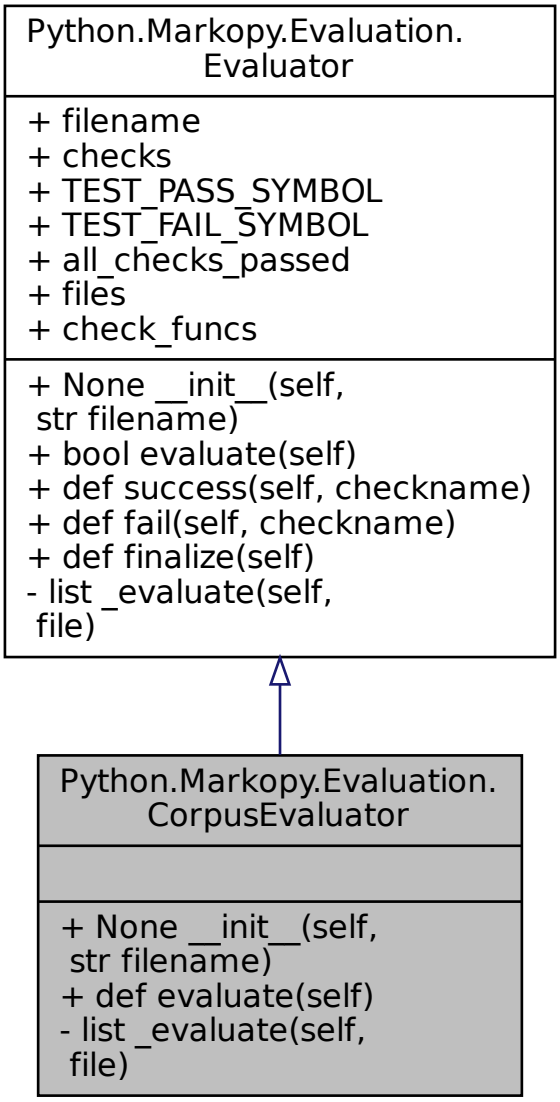
8.8 Python.Markopy.Evaluation.CorpusEvaluator Class Reference

evaluate a corpus

Inheritance diagram for Python.Markopy.Evaluation.CorpusEvaluator:



Collaboration diagram for Python.Markopy.Evaluation.CorpusEvaluator:



Public Member Functions

- None __init__ (self, str filename)

  default constructor
- def evaluate (self)
- def success (self, checkname)

  pass a test
- def fail (self, checkname)

  fail a test
- def finalize (self)

Public Attributes

- filename
- checks
- TEST_PASS_SYMBOL
- TEST_FAIL_SYMBOL
- all_checks_passed
- files
- check_funcs

Private Member Functions

- list _evaluate (self, file)

  evaluate a single file.

8.8.1 Detailed Description

evaluate a corpus
Definition at line 238 of file evaluate.py.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 __init__()

```
None Python.Markopy.Evaluation.CorpusEvaluator.__init__ (
              self,
             str filename )
```

default constructor

Parameters

filename	filename or pattern to check

Reimplemented from Python.Markopy.Evaluation.Evaluator.
Definition at line 244 of file evaluate.py.

```
00244     def __init__(self, filename: str) -> None:
00245         """!
00246         @brief default constructor
00247         @param filename filename or pattern to check
00248         """
00249         valid = super().__init__(filename)
00250         if not valid:
00251             return False
00252
```

8.8.3 Member Function Documentation

8.8.3.1 _evaluate()

```
list Python.Markopy.Evaluation.CorpusEvaluator._evaluate (
              self,
              file )  [private]
```

evaluate a single file.
Remove reading file because it should be read line by line.

Parameters

file	corpus filename to evaluate

Reimplemented from Python.Markopy.Evaluation.Evaluator.
Definition at line 295 of file evaluate.py.

```
00295     def _evaluate(self, file) -> list:
00296         """!
00297         @brief evaluate a single file. Remove reading file because it should be read line by line.
00298         @param file corpus filename to evaluate
00299         """
00300         if(not os.path.isfile(file)):
00301             logging.pprint(f"Given file {file} is not a valid filename")
00302             return False
00303         else:
00304             return True
00305
```

Referenced by Python.Markopy.Evaluation.Evaluator.evaluate().
Here is the caller graph for this function:

8.8.3.2 evaluate()

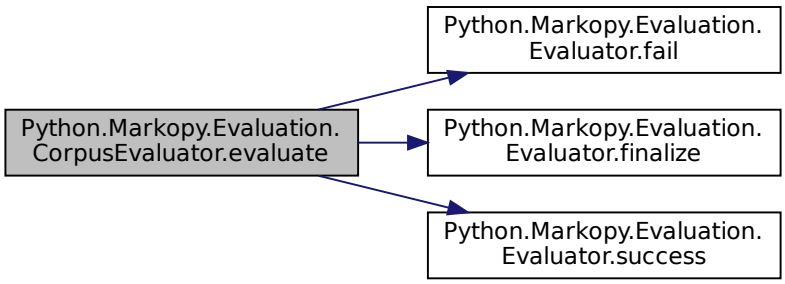
```
def Python.Markopy.Evaluation.CorpusEvaluator.evaluate (
            self )
```

Reimplemented from Python.Markopy.Evaluation.Evaluator.
Definition at line 253 of file evaluate.py.

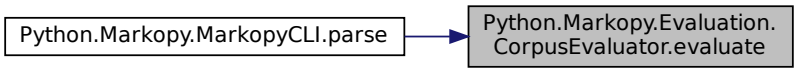
```
00253     def evaluate(self):
00254         "! @brief evalute a corpus. Might take a long time"
00255         logging.pprint("WARNING: This takes a while with larger corpus files", 2)
00256         logging.VERBOSITY=2
00257         logging.SHOW_STACK_THRESHOLD=3
00258         super().evaluate()
00259         for file in self.files:
00260
00261             delimiter = ''
00262             sum=0
00263             max=0
00264             total_chars = 0
00265             lines_count = 0
00266             bDelimiterConflict=False
00267             logging.pprint(f"Corpus: {file.split('/')[-1]}: ",2)
00268             with open(file, "rb") as corpus:
00269                 for line in corpus:
00270                     lines_count+=1
00271                     match = re.match(r"([0-9]+)(.)(.*)\n", line.decode()).groups()
00272                     if(delimiter and delimiter!=match[1]):
00273                         bDelimiterConflict = True
00274
00275                     elif(not delimiter):
00276                         delimiter = match[1]
00277                         logging.pprint(f"Delimiter is: {delimiter.encode()}")
00278                     sum +=int(match[0])
00279                     total_chars += len(match[2])
00280                     if(int(match[0])>max):
00281                         max=int(match[0])
00282
00283                 if(bDelimiterConflict):
00284                     self.fail("Incorrect delimiter found")
00285                 else:
00286                     self.success("No structural conflicts")
00287
00288                 logging.pprint(f"Total number of lines: {lines_count}")
00289                 logging.pprint(f"Sum of all string weights: {sum}")
00290                 logging.pprint(f"Character total: {total_chars}")
00291                 logging.pprint(f"Average length: {total_chars/lines_count}")
00292                 logging.pprint(f"Average weight: {sum/lines_count}")
00293
00294             self.finalize()
```

References Python.Markopy.Evaluation.Evaluator.fail(), Python.Markopy.Evaluation.Evaluator.files, Python.Markopy.Evaluation.Evalua
and Python.Markopy.Evaluation.Evaluator.success().
Referenced by Python.Markopy.MarkopyCLI.parse().

Here is the call graph for this function:



Here is the caller graph for this function:



8.8.3.3 fail()

```
def Python.Markopy.Evaluation.Evaluator.fail (
              self,
              checkname )  [inherited]
```

fail a test

Parameters

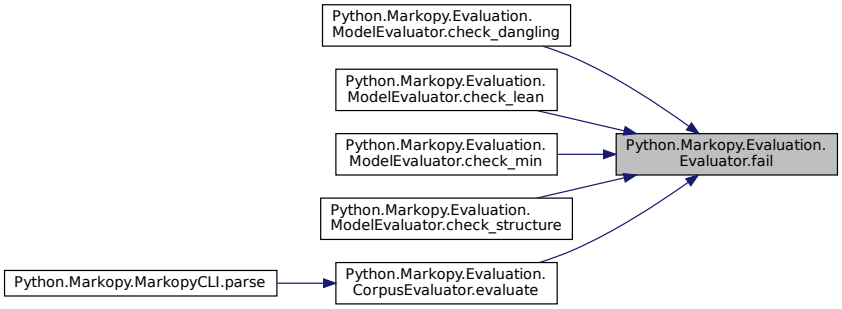
checkname	text to display with the check

Definition at line 72 of file evaluate.py.

```
00072     def fail(self, checkname):
00073         """!
00074         @brief fail a test
00075         @param checkname text to display with the check
00076         """
00077 
00078         self.all_checks_passed = False
00079         self.checks.append((checkname, self.TEST_FAIL_SYMBOL))
00080 
```

References Python.Markopy.Evaluation.Evaluator.all_checks_passed, Python.Markopy.Evaluation.Evaluator.checks, and Python.Markopy.Evaluation.Evaluator.TEST_FAIL_SYMBOL.
Referenced by Python.Markopy.Evaluation.ModelEvaluator.check_dangling(), Python.Markopy.Evaluation.ModelEvaluator.check_lean Python.Markopy.Evaluation.ModelEvaluator.check_min(), Python.Markopy.Evaluation.ModelEvaluator.check_structure(), and Python.Markopy.Evaluation.CorpusEvaluator.evaluate().

Here is the caller graph for this function:



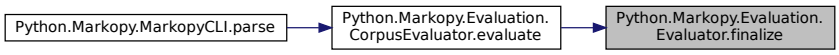
8.8.3.4 finalize()

```
def Python.Markopy.Evaluation.Evaluator.finalize (
              self )  [inherited]
```

Definition at line 81 of file evaluate.py.

```
00081     def finalize(self):
00082         "! @brief finalize an evaluation and print checks"
00083         print("\n################ Checks ################ ")
00084         for test in self.checks:
00085             logging.pprint(f"{test[0]:30}:{test[1]} ")
00086         print("\n")
00087         self.checks = []
00088         return self.all_checks_passed
00089
```

References Python.Markopy.Evaluation.Evaluator.all_checks_passed, and Python.Markopy.Evaluation.Evaluator.checks.
Referenced by Python.Markopy.Evaluation.CorpusEvaluator.evaluate().
Here is the caller graph for this function:



8.8.3.5 success()

```
def Python.Markopy.Evaluation.Evaluator.success (
              self,
              checkname )  [inherited]
```

pass a test

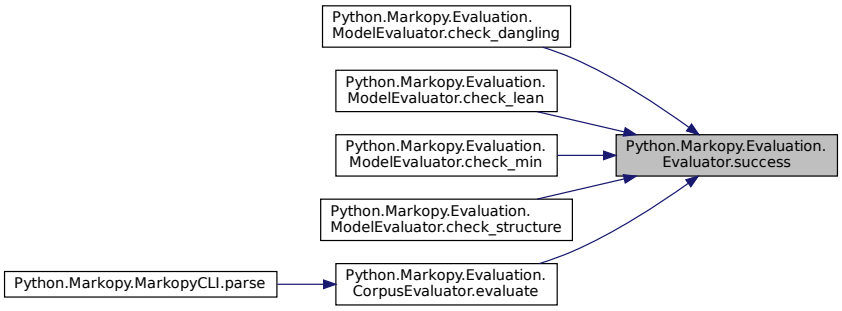
Parameters

checkname	text to display with the check

Definition at line 65 of file evaluate.py.

```
00065     def success(self, checkname):
00066         """!
00067         @brief pass a test
00068         @param checkname text to display with the check
00069         """
00070         self.checks.append((checkname, self.TEST_PASS_SYMBOL))
00071
```

References Python.Markopy.Evaluation.Evaluator.checks, and Python.Markopy.Evaluation.Evaluator.TEST_PASS_SYMBOL.
Referenced by Python.Markopy.Evaluation.ModelEvaluator.check_dangling(), Python.Markopy.Evaluation.ModelEvaluator.check_lean
Python.Markopy.Evaluation.ModelEvaluator.check_min(), Python.Markopy.Evaluation.ModelEvaluator.check_structure(),
and Python.Markopy.Evaluation.CorpusEvaluator.evaluate().
Here is the caller graph for this function:



8.8.4 Member Data Documentation

8.8.4.1 all_checks_passed

`Python.Markopy.Evaluation.Evaluator.all_checks_passed [inherited]`
Definition at line 36 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.fail(), and Python.Markopy.Evaluation.Evaluator.finalize().

8.8.4.2 check_funcs

`Python.Markopy.Evaluation.Evaluator.check_funcs [inherited]`
Definition at line 49 of file evaluate.py.

8.8.4.3 checks

`Python.Markopy.Evaluation.Evaluator.checks [inherited]`
Definition at line 33 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.fail(), Python.Markopy.Evaluation.Evaluator.finalize(), and Python.Markopy.Evaluation.Evaluator.success().

8.8.4.4 filename

`Python.Markopy.Evaluation.Evaluator.filename [inherited]`
Definition at line 32 of file evaluate.py.

8.8.4.5 files

`Python.Markopy.Evaluation.Evaluator.files [inherited]`
Definition at line 37 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.evaluate(), Python.Markopy.Evaluation.ModelEvaluator.evaluate(),
and Python.Markopy.Evaluation.CorpusEvaluator.evaluate().

8.8.4.6 TEST_FAIL_SYMBOL

`Python.Markopy.Evaluation.Evaluator.TEST_FAIL_SYMBOL [inherited]`
Definition at line 35 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.fail().

8.8.4.7 TEST_PASS_SYMBOL

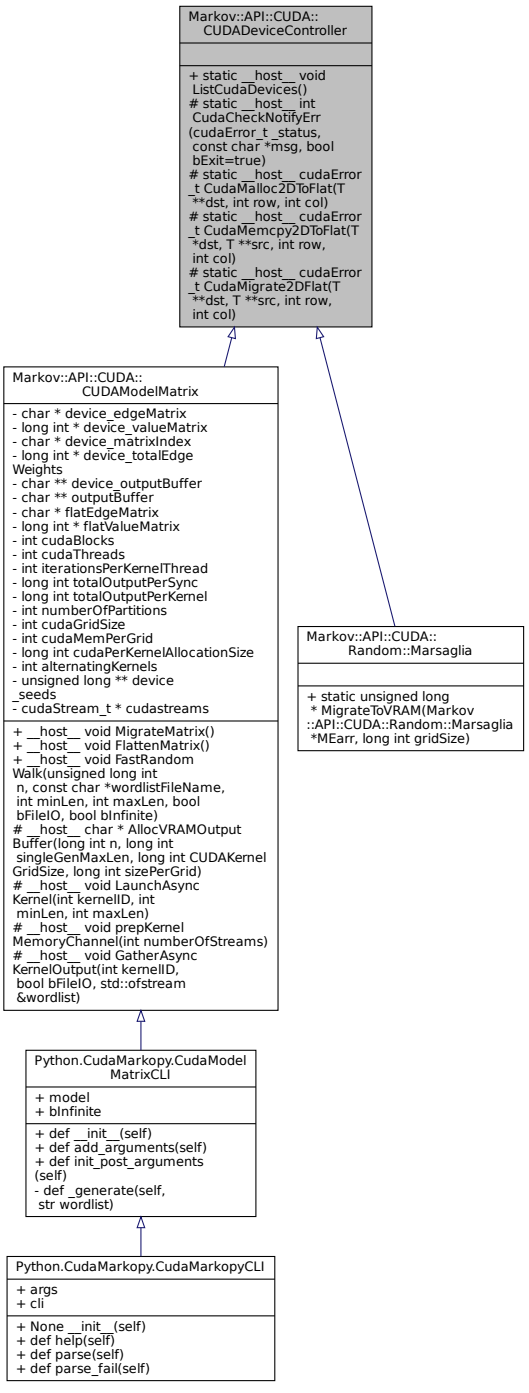
`Python.Markopy.Evaluation.Evaluator.TEST_PASS_SYMBOL [inherited]`
Definition at line 34 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.success().
The documentation for this class was generated from the following file:

- Markopy/Markopy/src/CLI/evaluate.py

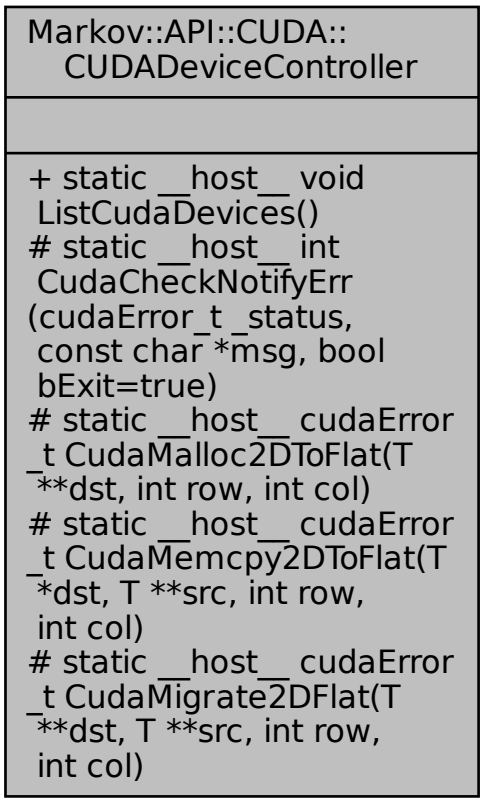
8.9 Markov::API::CUDA::CUDADeviceController Class Reference

Controller class for CUDA device.
`#include <cudaDeviceController.h>`

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



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



Static Public Member Functions

- static __host__ void ListCudaDevices ()

  List CUDA devices in the system.

Static Protected Member Functions

- static __host__ int CudaCheckNotifyErr (cudaError_t _status, const char ∗msg, bool bExit=true)

  Check results of the last operation on GPU.

- template<typename T >
  static __host__ cudaError_t CudaMalloc2DToFlat (T ∗∗dst, int row, int col)

  Malloc a 2D array in device space.

- template<typename T >
  static __host__ cudaError_t CudaMemcpy2DToFlat (T ∗dst, T ∗∗src, int row, int col)

  Memcpy a 2D array in device space after flattening.

- template<typename T >
  static __host__ cudaError_t CudaMigrate2DFlat (T ∗∗dst, T ∗∗src, int row, int col)

  Both malloc and memcpy a 2D array into device VRAM.

8.9.1 Detailed Description

Controller class for CUDA device.

This implementation only supports Nvidia devices.
Definition at line 18 of file cudaDeviceController.h.

8.9.2 Member Function Documentation

8.9.2.1 CudaCheckNotifyErr()

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

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

Parameters

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

Returns

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

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

Definition at line 32 of file cudaDeviceController.cu.

```
00032
                    {
00033         if (_status != cudaSuccess) {
00034             std::cerr « "\033[1;31m" « msg  « " -> " « cudaGetErrorString(_status)  « " ("« _status «
       ")" « "\033[0m" « "\n";
00035
00036             if(bExit) {
00037                 cudaDeviceReset();
00038                 exit(1);
00039             }
00040         }
00041         return 0;
00042     }
```

8.9.2.2 CudaMalloc2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat (
            T ** dst,
            int row,
            int col )  [inline], [static], [protected]
```

Malloc a 2D array in device space.
This function will allocate enough space on VRAM for flattened 2D array.

Parameters

dst	destination pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

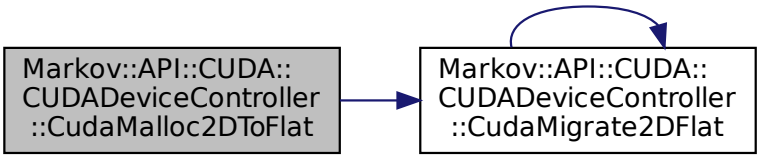
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess){
    CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
}
```

Definition at line 75 of file cudaDeviceController.h.

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

References CudaMigrate2DFlat().

Here is the call graph for this function:



8.9.2.3 CudaMemcpy2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat (
            T * dst,
            T ** src,
            int row,
            int col )  [inline], [static], [protected]
```

Memcpy a 2D array in device space after flattening.

Resulting buffer will not be true 2D array.

Parameters

dst	destination pointer
rc	source pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

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

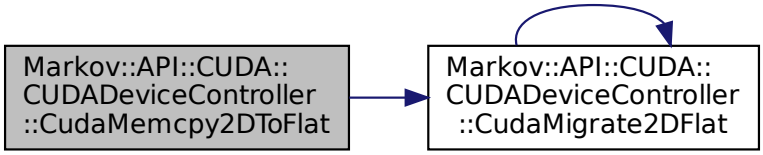
Definition at line 103 of file cudaDeviceController.h.

```
00103                                                                                {
```

```
00104             T* tempbuf = new T[row*col];
00105             for(int i=0;i<row;i++){
00106                 memcpy(&(tempbuf[row*i]), src[i], col);
00107             }
00108             return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00109
00110         }
```

References CudaMigrate2DFlat().

Here is the call graph for this function:



8.9.2.4 CudaMigrate2DFlat()

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

Both malloc and memcpy a 2D array into device VRAM.

Resulting buffer will not be true 2D array.

Parameters

dst	destination pointer
rc	source pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

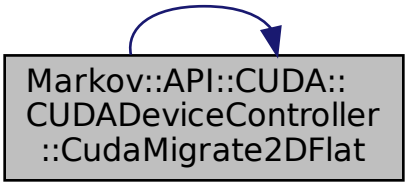
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMigrate2DFlat<long int>(
   &dst, this->valueMatrix, this->matrixSize, this->matrixSize);
CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
```

Definition at line 132 of file cudaDeviceController.h.

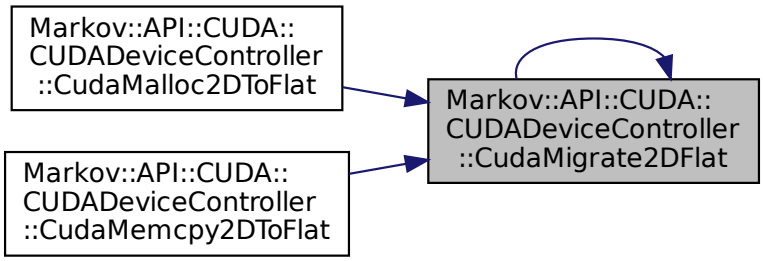
```
00132                                                                                 {
00133             cudaError_t cudastatus;
00134             cudastatus = CudaMalloc2DToFlat<T>(dst, row, col);
00135             if(cudastatus!=cudaSuccess){
00136                 CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
00137                 return cudastatus;
00138             }
00139             cudastatus = CudaMemcpy2DToFlat<T>(*dst,src,row,col);
00140             CudaCheckNotifyErr(cudastatus, "  CudaMemcpy2DToFlat Failed.", false);
00141             return cudastatus;
00142         }
```

References CudaMigrate2DFlat().

Referenced by CudaMalloc2DToFlat(), CudaMemcpy2DToFlat(), and CudaMigrate2DFlat().
Here is the call graph for this function:



Here is the caller graph for this function:



8.9.2.5 ListCudaDevices()

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

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

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

Definition at line 16 of file cudaDeviceController.cu.

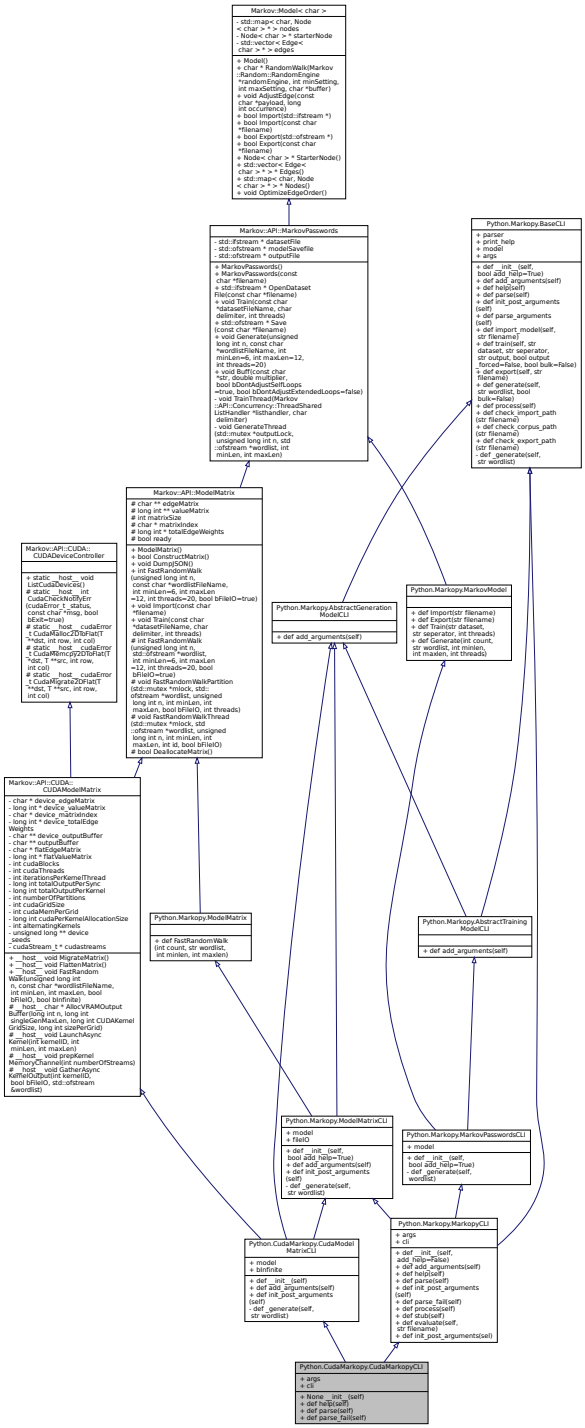
```
00016                                                       { //list cuda Capable devices on
       host.
00017         int nDevices;
00018         cudaGetDeviceCount(&nDevices);
00019         for (int i = 0; i < nDevices; i++) {
00020             cudaDeviceProp prop;
00021             cudaGetDeviceProperties(&prop, i);
00022             std::cerr « "Device Number: " «  i  « "\n";
00023             std::cerr « "Device name: " « prop.name « "\n";
00024             std::cerr « "Memory Clock Rate (KHz): " « prop.memoryClockRate « "\n";
00025             std::cerr « "Memory Bus Width (bits): " « prop.memoryBusWidth « "\n";
00026             std::cerr « "Peak Memory Bandwidth (GB/s): " « 2.0 * prop.memoryClockRate *
       (prop.memoryBusWidth / 8) / 1.0e6 « "\n";
00027             std::cerr « "Max Linear Threads: " « prop.maxThreadsDim[0] « "\n";
00028
00029         }
00030     }
```

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

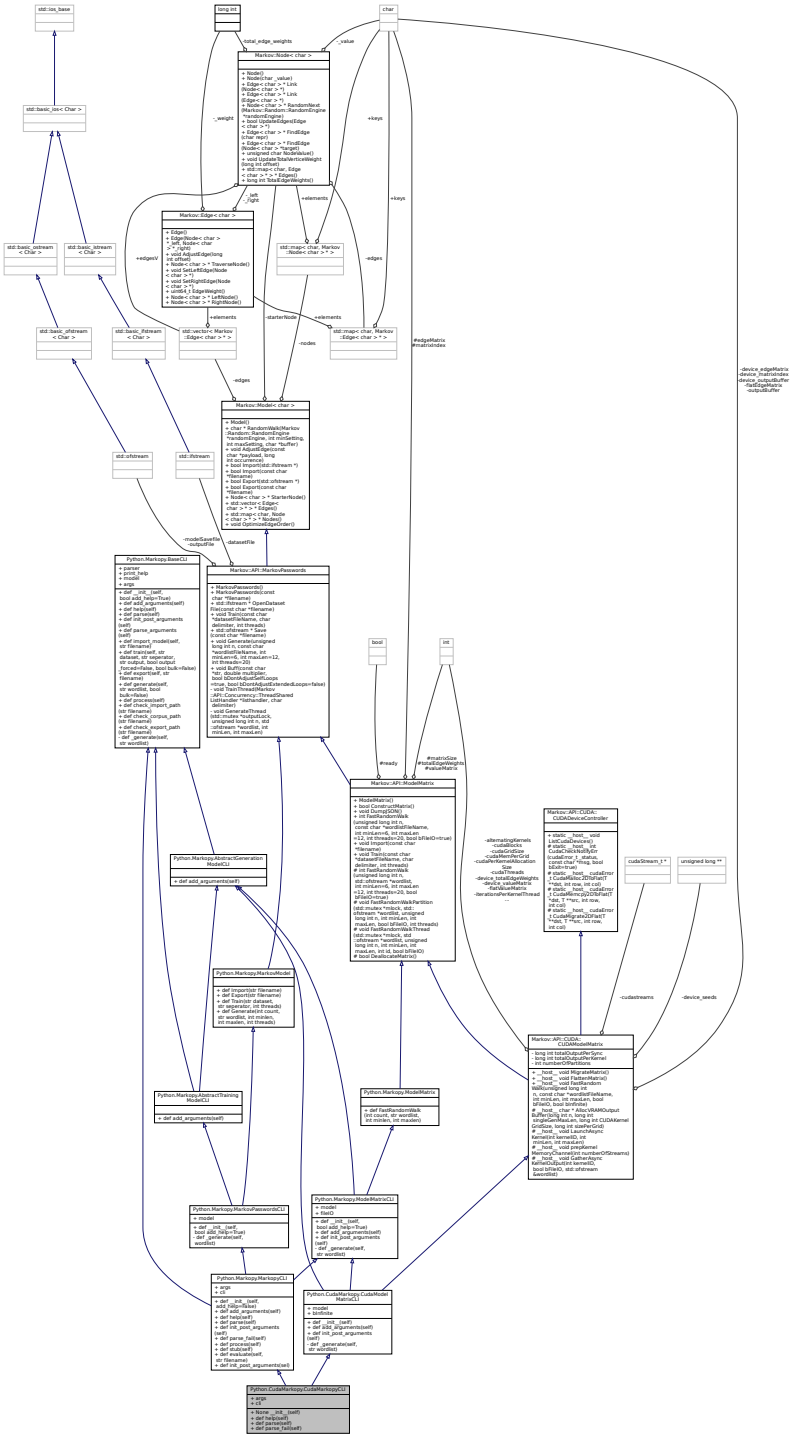
- Markopy/CudaMarkovAPI/src/cudaDeviceController.h
- Markopy/CudaMarkovAPI/src/cudaDeviceController.cu

8.10 Python.CudaMarkopy.CudaMarkopyCLI Class Reference

CUDA extension to MarkopyCLI.
Inheritance diagram for Python.CudaMarkopy.CudaMarkopyCLI:



Collaboration diagram for Python.CudaMarkopy.CudaMarkopyCLI:



Public Member Functions

- None __init__ (self)
- def help (self)

  overload help function to print submodel helps
- def parse (self)
- def parse_fail (self)
- def add_arguments (self)

add -mt/–model_type constructor

- def init_post_arguments (self)
- def init_post_arguments (sel)
- def process (self)

  Process parameters for operation.
- def stub (self)
- def evaluate (self, str filename)
- def parse_arguments (self)
- def parse_arguments (self)
- def parse_arguments (self)
- def parse_arguments (self)
- def import_model (self, str filename)

  Import a model file.
- def import_model (self, str filename)

  Import a model file.
- def import_model (self, str filename)

  Import a model file.
- def import_model (self, str filename)

  Import a model file.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def export (self, str filename)

  Export model to a file.
- def export (self, str filename)

  Export model to a file.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def FastRandomWalk (int count, str wordlist, int minlen, int maxlen)
- int FastRandomWalk (unsigned long int n, const char ∗wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

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

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

  Debug function to dump the model to a JSON file.

- void Import (const char *filename)

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

  Import a file to construct the model.
- def Import (str filename)
- bool Import (std::ifstream *)

  Import a file to construct the model.
- void Train (const char *datasetFileName, char delimiter, int threads)

  Train the model with the dataset file.
- def Train (str dataset, str seperator, int threads)
- std::ifstream * OpenDatasetFile (const char *filename)

  Open dataset file and return the ifstream pointer.
- std::ifstream * OpenDatasetFile (const char *filename)

  Open dataset file and return the ifstream pointer.
- std::ofstream * Save (const char *filename)

  Export model to 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.
- def Generate (int count, str wordlist, int minlen, int maxlen, int threads)
- void Generate (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

  Call Markov::Model::RandomWalk n times, and collect output.
- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- char * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

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

  Adjust the model with a single string.
- bool Export (std::ofstream *)

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

  Open a file to export with filename, and call bool Model::Export with std::ofstream.
- def Export (str filename)
- 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.

- Node< char > * StarterNode ()

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

  Return a vector of all the edges in the model.
- std::vector< Edge< char > * > * Edges ()

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

  Return starter Node.
- std::map< char, Node< char > * > * Nodes ()

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.
- def add_arguments (self)
- def init_post_arguments (self)
- def parse_arguments (self)
- def parse_arguments (self)
- def import_model (self, str filename)

  Import a model file.
- def import_model (self, str filename)

  Import a model file.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def FastRandomWalk (int count, str wordlist, int minlen, int maxlen)
- int FastRandomWalk (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- __host__ void FastRandomWalk (unsigned long int n, const char *wordlistFileName, int minLen, int maxLen, bool bFileIO, bool bInfinite)

  Random walk on the Matrix-reduced Markov::Model.
- int FastRandomWalk (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

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

  Construct the related Matrix data for the model.
- bool ConstructMatrix ()

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

  Debug function to dump the model to a JSON file.
- void DumpJSON ()

Debug function to dump the model to a JSON file.

- void Import (const char *filename)

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

  Import a file to construct the model.
- void Import (const char *filename)

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

  Import a file to construct the model.
- void Train (const char *datasetFileName, char delimiter, int threads)

  Train the model with the dataset file.
- void Train (const char *datasetFileName, char delimiter, int threads)

  Train the model with the dataset file.
- std::ifstream * OpenDatasetFile (const char *filename)

  Open dataset file and return the ifstream pointer.
- std::ifstream * OpenDatasetFile (const char *filename)

  Open dataset file and return the ifstream pointer.
- std::ofstream * Save (const char *filename)

  Export model to file.
- std::ofstream * Save (const char *filename)

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

  Call Markov::Model::RandomWalk n times, and collect output.
- void Generate (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

  Call Markov::Model::RandomWalk n times, and collect output.
- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- char * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

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

  Adjust the model with a single string.
- bool Export (std::ofstream *)

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

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

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

  Return a vector of all the edges in the model.
- std::vector< Edge< char > * > * Edges ()

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

  Return starter Node.
- std::map< char, Node< char > * > * Nodes ()

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.
- __host__ void MigrateMatrix ()

  Migrate the class members to the VRAM.
- __host__ void FlattenMatrix ()

  Flatten migrated matrix from 2d to 1d.

Static Public Member Functions

- def check_import_path (str filename)

  check import path for validity
- def check_import_path (str filename)

  check import path for validity
- def check_import_path (str filename)

  check import path for validity
- def check_import_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- def check_import_path (str filename)

  check import path for validity
- def check_import_path (str filename)

  check import path for validity

- def check_corpus_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- static __host__ void ListCudaDevices ()

  List CUDA devices in the system.

Public Attributes

- args
- cli
- parser
- parser
- parser
- parser
- print_help
- print_help
- print_help
- print_help
- model
- model
- model
- fileIO
- model
- bInfinite
- fileIO
- parser
- parser
- print_help
- print_help

Protected Member Functions

- int FastRandomWalk (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- void FastRandomWalkPartition (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

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

  A single thread of a single partition of FastRandomWalk.
- bool DeallocateMatrix ()

  Deallocate matrix and make it ready for re-construction.
- int FastRandomWalk (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- int FastRandomWalk (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

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

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

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

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

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

  A single thread of a single partition of FastRandomWalk.
- bool DeallocateMatrix ()

  Deallocate matrix and make it ready for re-construction.
- bool DeallocateMatrix ()

  Deallocate matrix and make it ready for re-construction.
- __host__ char * AllocVRAMOutputBuffer (long int n, long int singleGenMaxLen, long int CUDAKernelGrid↩ Size, long int sizePerGrid)

  Allocate the output buffer for kernel operation.
- __host__ void LaunchAsyncKernel (int kernelID, int minLen, int maxLen)
- __host__ void prepKernelMemoryChannel (int numberOfStreams)
- __host__ void GatherAsyncKernelOutput (int kernelID, bool bFileIO, std::ofstream &wordlist)

Static Protected Member Functions

- static __host__ int CudaCheckNotifyErr (cudaError_t _status, const char *msg, bool bExit=true)

  Check results of the last operation on GPU.
- template<typename T >
  static __host__ cudaError_t CudaMalloc2DToFlat (T **dst, int row, int col)

  Malloc a 2D array in device space.
- template<typename T >
  static __host__ cudaError_t CudaMemcpy2DToFlat (T *dst, T **src, int row, int col)

  Memcpy a 2D array in device space after flattening.
- template<typename T >
  static __host__ cudaError_t CudaMigrate2DFlat (T **dst, T **src, int row, int col)

  Both malloc and memcpy a 2D array into device VRAM.

Protected Attributes

- char ** edgeMatrix

  2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** valueMatrix

  2-d Integer array for the value Matrix (For the weights of Edges)
- int matrixSize

  to hold Matrix size
- char * matrixIndex

  to hold the Matrix index (To hold the orders of 2-D arrays')
- long int * totalEdgeWeights

  Array of the Total Edge Weights.
- bool ready

  True when matrix is constructed. False if not.
- char ** edgeMatrix

  2-D Character array for the edge Matrix (The characters of Nodes)

- char ** edgeMatrix

  2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** valueMatrix

  2-d Integer array for the value Matrix (For the weights of Edges)
- long int ** valueMatrix

  2-d Integer array for the value Matrix (For the weights of Edges)
- int matrixSize

  to hold Matrix size
- int matrixSize

  to hold Matrix size
- char * matrixIndex

  to hold the Matrix index (To hold the orders of 2-D arrays')
- char * matrixIndex

  to hold the Matrix index (To hold the orders of 2-D arrays')
- long int * totalEdgeWeights

  Array of the Total Edge Weights.
- long int * totalEdgeWeights

  Array of the Total Edge Weights.
- bool ready

  True when matrix is constructed. False if not.
- bool ready

  True when matrix is constructed. False if not.

Private Member Functions

- def _generate (self, str wordlist)

  wrapper for generate function.
- 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

  Dataset file input of our system
- std::ofstream * outputFile

  File to save model of our system
- std::map< char, Node< char > * > nodes

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

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

  A list of all edges in this model.
- char * device_edgeMatrix

  VRAM Address pointer of edge matrix (from modelMatrix.h)
- long int * device_valueMatrix

  VRAM Address pointer of value matrix (from modelMatrix.h)

- char * device_matrixIndex

  VRAM Address pointer of matrixIndex (from modelMatrix.h)
- long int * device_totalEdgeWeights

  VRAM Address pointer of total edge weights (from modelMatrix.h)
- char ** device_outputBuffer

  RandomWalk results in device.
- char ** outputBuffer

  RandomWalk results in host.
- char * flatEdgeMatrix

  Adding Edge matrix end-to-end and resize to 1-D array for better perfomance on traversing.
- long int * flatValueMatrix

  Adding Value matrix end-to-end and resize to 1-D array for better perfomance on traversing.
- int cudaBlocks
- int cudaThreads
- int iterationsPerKernelThread
- long int totalOutputPerSync
- long int totalOutputPerKernel
- int numberOfPartitions
- int cudaGridSize
- int cudaMemPerGrid
- long int cudaPerKernelAllocationSize
- int alternatingKernels
- unsigned long ** device_seeds
- cudaStream_t * cudastreams

8.10.1 Detailed Description

CUDA extension to MarkopyCLI.
Adds CudaModelMatrixCLI to the interface.
Definition at line 46 of file cudamarkopy.py.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 __init__()

```
None Python.CudaMarkopy.CudaMarkopyCLI.__init__ (
                self )
```

Reimplemented from Python.CudaMarkopy.CudaModelMatrixCLI.
Definition at line 53 of file cudamarkopy.py.

```
00053     def __init__(self) -> None:
00054         "! @brief initialize CLI selector"
00055         markopy.MarkopyCLI.__init__(self,add_help=False)
00056         self.parser.epilog+=f"""
00057         {__file__.split("/")[-1]} -mt CUDA generate trained.mdl -n 500 -w output.txt
00058             Import trained.mdl, and generate 500 lines to output.txt
00059         """
00060
```

References Python.Markopy.BaseCLI.parser.

8.10.3 Member Function Documentation

8.10.3.1 _generate()

```
def Python.Markopy.BaseCLI._generate (
              self,
             str wordlist )  [private], [inherited]
```

wrapper for generate function.
This can be overloaded by other models

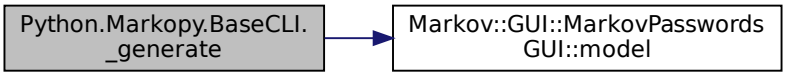
Parameters

wordlist	filename to generate to

Reimplemented in Python.Markopy.ModelMatrixCLI, and Python.CudaMarkopy.CudaModelMatrixCLI.
Definition at line 161 of file base.py.

```
00161     def _generate(self, wordlist : str):
00162         """!
00163         @brief wrapper for generate function. This can be overloaded by other models
00164         @param wordlist filename to generate to
00165         """
00166         self.model.Generate(int(self.args.count), wordlist, int(self.args.min), int(self.args.max),
      int(self.args.threads))
00167
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.generate().
Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.2 add_arguments() [1/2]

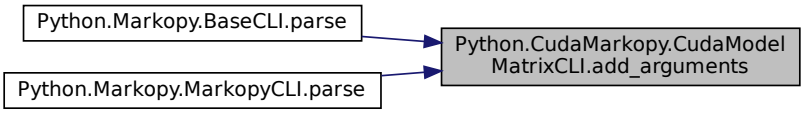
```
def Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments (
              self )  [inherited]
```

Reimplemented from Python.Markopy.AbstractGenerationModelCLI.
Definition at line 67 of file cudammx.py.

```
00067     def add_arguments(self):
00068         super().add_arguments()
00069         self.parser.add_argument("-if", "--infinite", action="store_true", help="Infinite generation
      mode")
00070
```

References Python.Markopy.BaseCLI.parser.
Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.10.3.3 add_arguments() [2/2]

```
def Python.Markopy.MarkopyCLI.add_arguments (
              self )  [inherited]
```

add -mt/–model_type constructor

Reimplemented from Python.Markopy.BaseCLI.

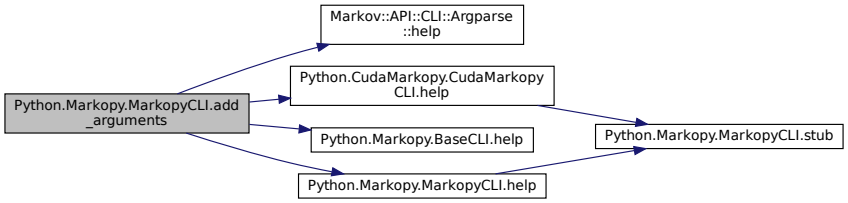
Definition at line 83 of file markopy.py.

```
00083     def add_arguments(self):
00084         """!
00085         @brief add -mt/--model_type constructor
00086         """
00087         self.parser.add_argument("-mt", "--model_type", default="_MMX", help="Model type to use.
      Accepted values: MP, MMX")
00088         self.parser.add_argument("-h", "--help", action="store_true", help="Model type to use.
      Accepted values: MP, MMX")
00089         self.parser.add_argument("-ev", "--evaluate", help="Evaluate a models integrity")
00090         self.parser.add_argument("-evt", "--evaluate_type", help="Evaluation type, model or corpus")
00091         self.parser.print_help = self.help
00092
```

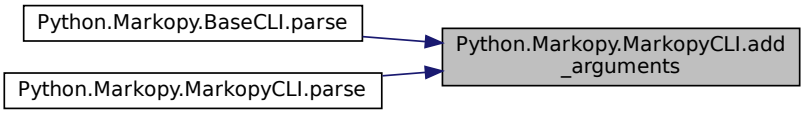
References Markov::API::CLI::Argparse.help(), Python.CudaMarkopy.CudaMarkopyCLI.help(), Python.Markopy.BaseCLI.help(), Python.Markopy.MarkopyCLI.help(), and Python.Markopy.BaseCLI.parser.

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.4 AdjustEdge() [1/4]

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

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

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

Parameters

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

Definition at line 109 of file model.h.

```
00337                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.10.3.5 AdjustEdge() [2/4]

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

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

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

Parameters

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

Definition at line 109 of file model.h.

```
00337                                                                                                {
00338     NodeStorageType p = payload[0];
```

```
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.10.3.6 AdjustEdge() [3/4]

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

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

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

Parameters

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

Definition at line 109 of file model.h.

```
00337                                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.10.3.7 AdjustEdge() [4/4]

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

Adjust the model with a single string.
Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.
Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.
Example Use: Create an empty model and train it with string: "testdata"

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

Parameters

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

Definition at line 109 of file model.h.

```
00337                                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.10.3.8 AllocVRAMOutputBuffer()

```
__host__ char* Markov::API::CUDA::CUDAModelMatrix::AllocVRAMOutputBuffer (
            long int n,
            long int singleGenMaxLen,
            long int CUDAKernelGridSize,
            long int sizePerGrid )  [protected], [inherited]
```

Allocate the output buffer for kernel operation.
TODO

Parameters

n	- Number of passwords to generate.
singleGenMaxLen	- maximum string length for a single generation
CUDAKernelGridSize	- Total number of grid members in CUDA kernel
sizePerGrid	- Size to allocate per grid member

Returns

pointer to the allocation on VRAM

8.10.3.9 Buff() [1/4]

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

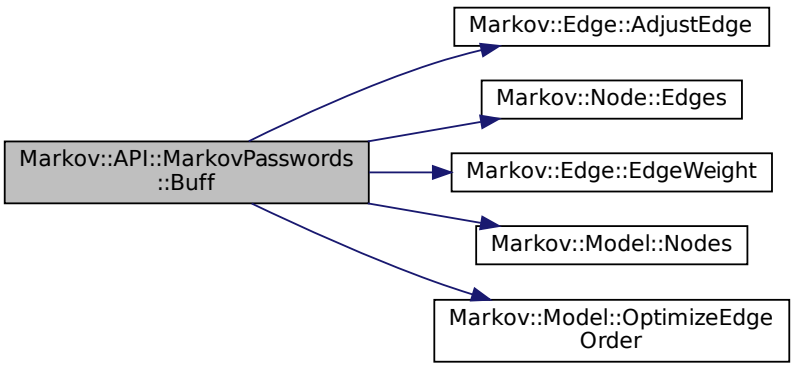
Definition at line 153 of file markovPasswords.cpp.

```
00153                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173 
00174         }
00175         i++;
00176     }
00177 
00178     this->OptimizeEdgeOrder();
00179 }
```

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

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.10 Buff() [2/4]

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

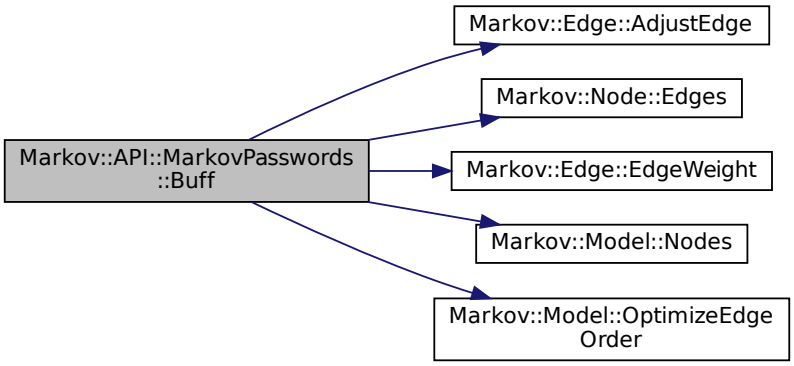
Definition at line 153 of file markovPasswords.cpp.

```
00153                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173
00174         }
00175         i++;
00176     }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

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

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.11 Buff() [3/4]

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file markovPasswords.cpp.

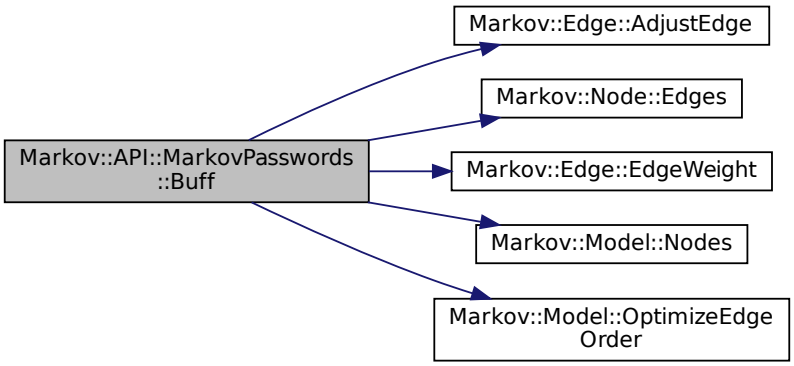
```
00153
                            {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
```

```
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173 
00174         }
00175         i++;
00176     }
00177 
00178     this->OptimizeEdgeOrder();
00179 }
```

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

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.12 Buff() [4/4]

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
```

```
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

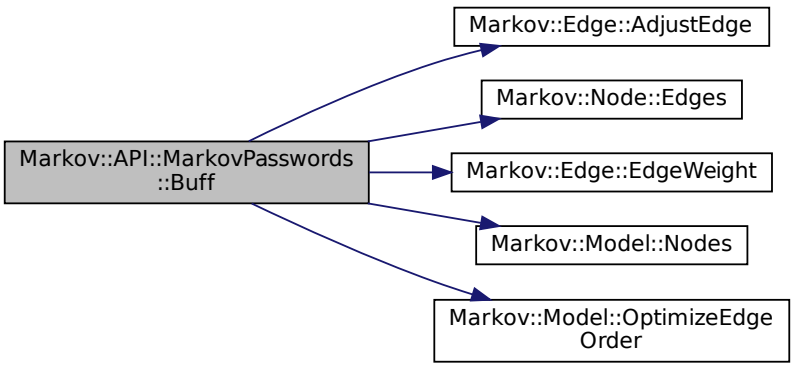
Definition at line 153 of file markovPasswords.cpp.

```
00153                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173
00174         }
00175         i++;
00176     }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

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

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.13 check_corpus_path() [1/6]

```
def Python.Markopy.BaseCLI.check_corpus_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186 
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.10.3.14 check_corpus_path() [2/6]

```
def Python.Markopy.BaseCLI.check_corpus_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
```

```
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.10.3.15 check_corpus_path() [3/6]

```
def Python.Markopy.BaseCLI.check_corpus_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.10.3.16 check_corpus_path() [4/6]

```
def Python.Markopy.BaseCLI.check_corpus_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
```

```
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.10.3.17 check_corpus_path() [5/6]

```
def Python.Markopy.BaseCLI.check_corpus_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.10.3.18 check_corpus_path() [6/6]

```
def Python.Markopy.BaseCLI.check_corpus_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
```

```
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.10.3.19 check_export_path() [1/6]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.10.3.20 check_export_path() [2/6]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
```

```
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.10.3.21 check_export_path() [3/6]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.10.3.22 check_export_path() [4/6]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
```

```
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.10.3.23 check_export_path() [5/6]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.10.3.24 check_export_path() [6/6]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
```

```
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.10.3.25 check_import_path() [1/6]

```
def Python.Markopy.BaseCLI.check_import_path (
             str filename )  [static], [inherited]
```

check import path for validity

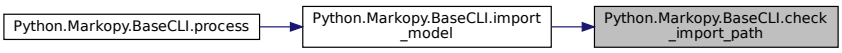
Parameters

filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:



8.10.3.26 check_import_path() [2/6]

```
def Python.Markopy.BaseCLI.check_import_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

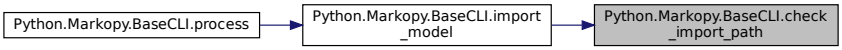
filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
```

```
00172         @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:



8.10.3.27 check_import_path() [3/6]

```
def Python.Markopy.BaseCLI.check_import_path (
            str filename )  [static], [inherited]
```

check import path for validity

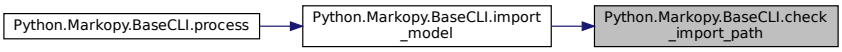
Parameters

filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:



8.10.3.28 check_import_path() [4/6]

```
def Python.Markopy.BaseCLI.check_import_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
```

```
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:

8.10.3.29 check_import_path() [5/6]

```
def Python.Markopy.BaseCLI.check_import_path (
             str filename )  [static], [inherited]
```

check import path for validity

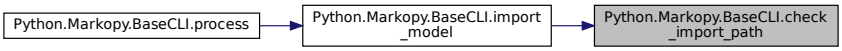
Parameters

filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:



8.10.3.30 check_import_path() [6/6]

```
def Python.Markopy.BaseCLI.check_import_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

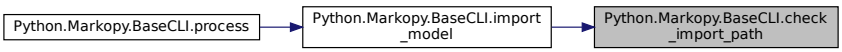
filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
```

Referenced by Python.Markopy.BaseCLI.import_model().

Here is the caller graph for this function:



8.10.3.31 ConstructMatrix() [1/3]

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

Construct the related Matrix data for the model.

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

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

Returns

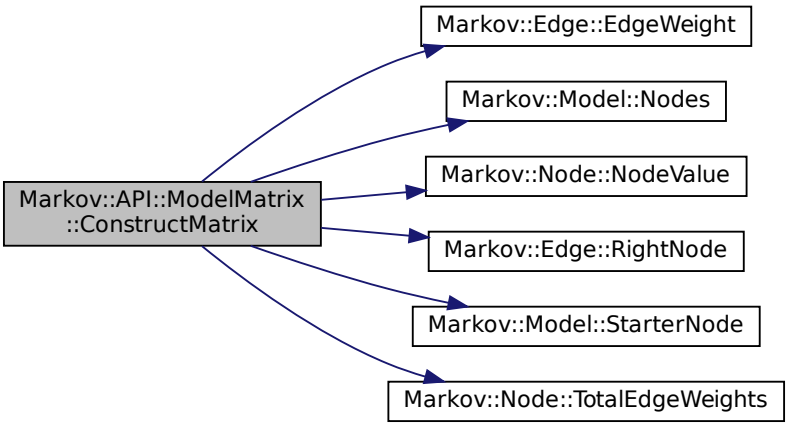
True if constructed. False if already construced.

Definition at line 31 of file modelMatrix.cpp.

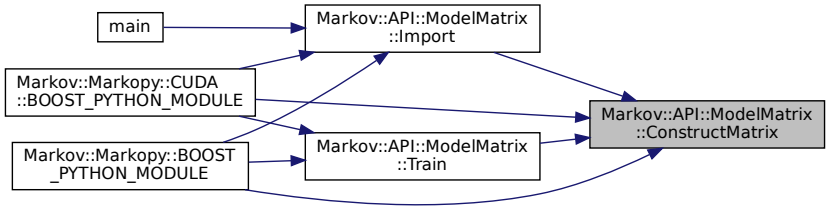
```
00031                                           {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034 
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037 
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
```

```
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072
00073         }
00074         i++;
00075     }
00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::API::ModelMatrix::n
Markov::API::ModelMatrix::matrixSize, Markov::Model< NodeStorageType >::Nodes(), Markov::Node< storageType >::NodeValue()
Markov::API::ModelMatrix::ready, Markov::Edge< NodeStorageType >::RightNode(), Markov::Model< NodeStorageType >::StarterN
Markov::API::ModelMatrix::totalEdgeWeights, Markov::Node< storageType >::TotalEdgeWeights(), and Markov::API::ModelMatrix::va
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(),
Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.32 ConstructMatrix() [2/3]

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

Construct the related Matrix data for the model.
This operation can be used after importing/training to allocate and populate the matrix content.
this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number

of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

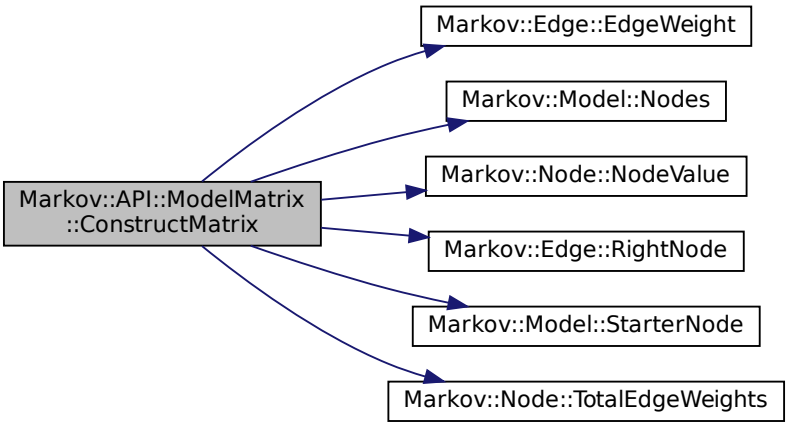
True if constructed. False if already construced.

Definition at line 31 of file modelMatrix.cpp.

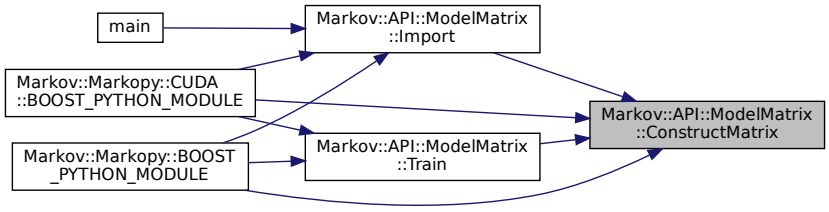
```
00031                                       {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034 
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037 
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072 
00073         }
00074         i++;
00075     }
00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::API::ModelMatrix::n
Markov::API::ModelMatrix::matrixSize, Markov::Model< NodeStorageType >::Nodes(), Markov::Node< storageType >::NodeValue()
Markov::API::ModelMatrix::ready, Markov::Edge< NodeStorageType >::RightNode(), Markov::Model< NodeStorageType >::StarterN
Markov::API::ModelMatrix::totalEdgeWeights, Markov::Node< storageType >::TotalEdgeWeights(), and Markov::API::ModelMatrix::va
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(),
Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.33 ConstructMatrix() [3/3]

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

Construct the related Matrix data for the model.
This operation can be used after importing/training to allocate and populate the matrix content.
this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

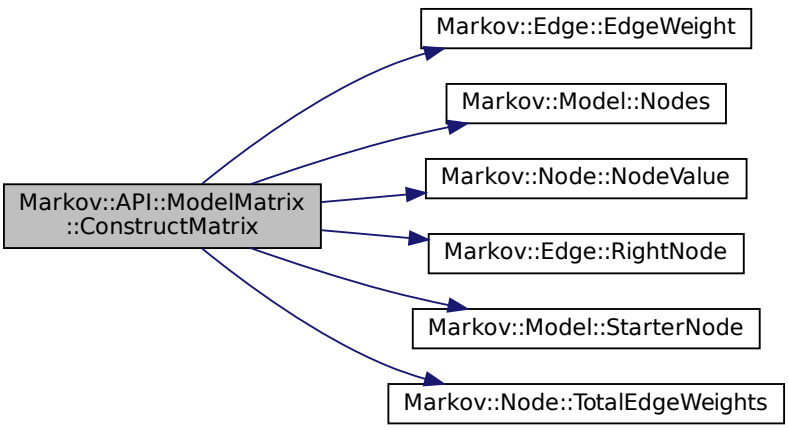
True if constructed. False if already construced.

Definition at line 31 of file modelMatrix.cpp.

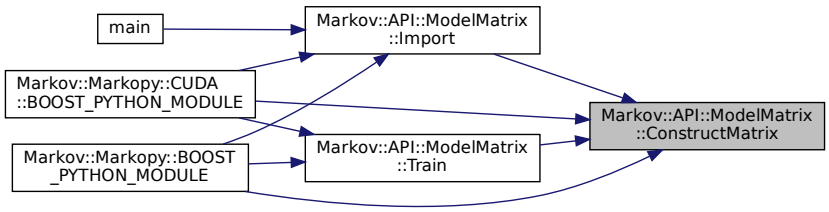
```
00031                                          {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034 
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037 
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
```

```
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072
00073         }
00074         i++;
00075     }
00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::API::ModelMatrix::
Markov::API::ModelMatrix::matrixSize, Markov::Model< NodeStorageType >::Nodes(), Markov::Node< storageType >::NodeValue()
Markov::API::ModelMatrix::ready, Markov::Edge< NodeStorageType >::RightNode(), Markov::Model< NodeStorageType >::StarterN
Markov::API::ModelMatrix::totalEdgeWeights, Markov::Node< storageType >::TotalEdgeWeights(), and Markov::API::ModelMatrix::va
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(),
Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.34 CudaCheckNotifyErr()

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

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

Parameters

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

Returns

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

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

Definition at line 32 of file cudaDeviceController.cu.

```
00032
                    {
00033         if (_status != cudaSuccess) {
00034             std::cerr « "\033[1;31m" « msg  « " -> " « cudaGetErrorString(_status)  « " ("« _status «
      ")" « "\033[0m" « "\n";
00035
00036             if(bExit) {
00037                 cudaDeviceReset();
00038                 exit(1);
00039             }
00040         }
00041         return 0;
00042     }
```

8.10.3.35 CudaMalloc2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat (
            T ** dst,
            int row,
            int col )  [inline], [static], [protected], [inherited]
```

Malloc a 2D array in device space.
This function will allocate enough space on VRAM for flattened 2D array.

Parameters

dst	destination pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

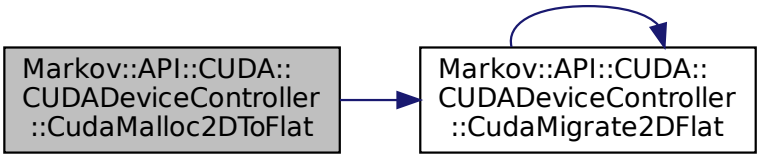
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess){
    CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
}
```

Definition at line 75 of file cudaDeviceController.h.

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

References Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().

Here is the call graph for this function:



8.10.3.36 CudaMemcpy2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat (
            T * dst,
            T ** src,
            int row,
            int col )  [inline], [static], [protected], [inherited]
```

Memcpy a 2D array in device space after flattening.

Resulting buffer will not be true 2D array.

Parameters

dst	destination pointer
rc	source pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

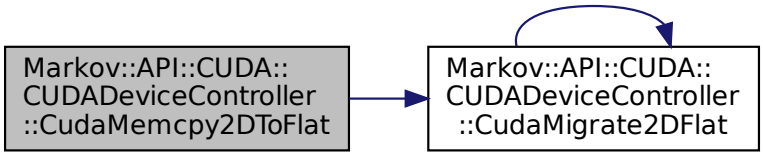
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
CudaCheckNotifyErr(cudastatus, "  CudaMemcpy2DToFlat Failed.", false);
```

Definition at line 103 of file cudaDeviceController.h.

```
00103                                                                                {
00104             T* tempbuf = new T[row*col];
00105             for(int i=0;i<row;i++){
00106                 memcpy(&(tempbuf[row*i]), src[i], col);
00107             }
00108             return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00109
00110         }
```

References Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().

Here is the call graph for this function:



8.10.3.37 CudaMigrate2DFlat()

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

Both malloc and memcpy a 2D array into device VRAM.

Resulting buffer will not be true 2D array.

Parameters

dst	destination pointer
rc	source pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

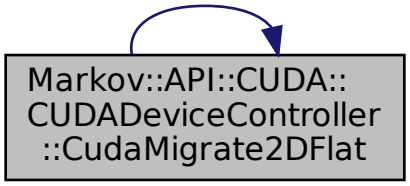
Example output:

```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMigrate2DFlat<long int>(
   &dst, this->valueMatrix, this->matrixSize, this->matrixSize);
CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
```

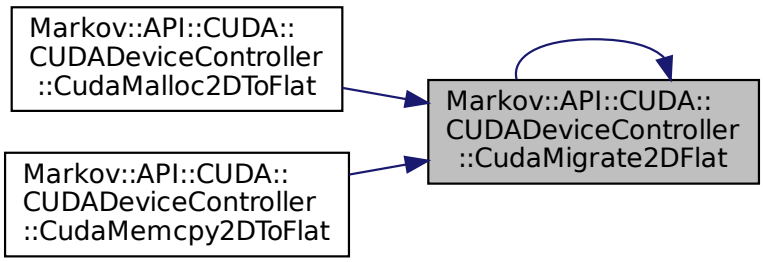
Definition at line 132 of file cudaDeviceController.h.

```
00132                                                                                                                {
00133             cudaError_t cudastatus;
00134             cudastatus = CudaMalloc2DToFlat<T>(dst, row, col);
00135             if(cudastatus!=cudaSuccess){
00136                 CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
00137                 return cudastatus;
00138             }
00139             cudastatus = CudaMemcpy2DToFlat<T>(*dst,src,row,col);
00140             CudaCheckNotifyErr(cudastatus, "  CudaMemcpy2DToFlat Failed.", false);
00141             return cudastatus;
00142         }
```

References Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().
Referenced by Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat(), Markov::API::CUDA::CUDADeviceController::Cu
and Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().
Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.38 DeallocateMatrix() [1/3]

`bool Markov::API::ModelMatrix::DeallocateMatrix ( )  [protected], [inherited]`

Deallocate matrix and make it ready for re-construction.

Returns

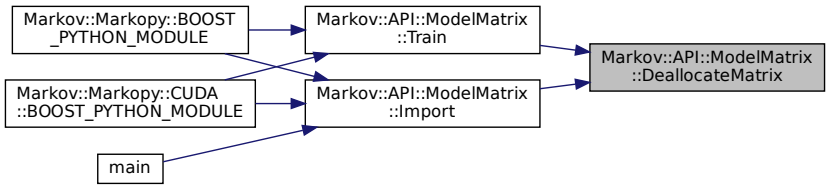
True if deallocated. False if matrix was not initialized

Definition at line 81 of file modelMatrix.cpp.

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

```
00085
00086     for(int i=0;i<this->matrixSize;i++){
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090
00091     for(int i=0;i<this->matrixSize;i++){
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
00099 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, Markov::API::ModelMatrix::ready, Markov::API::ModelMatrix::totalEdgeWeights, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().
Here is the caller graph for this function:



8.10.3.39 DeallocateMatrix() [2/3]

`bool Markov::API::ModelMatrix::DeallocateMatrix ( ) [protected], [inherited]`

Deallocate matrix and make it ready for re-construction.

Returns

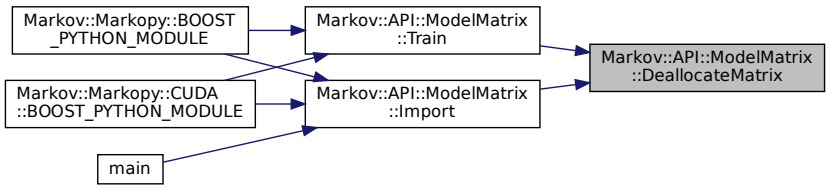
True if deallocated. False if matrix was not initialized

Definition at line 81 of file modelMatrix.cpp.

```
00081                                           {
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
00085
00086     for(int i=0;i<this->matrixSize;i++){
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090
00091     for(int i=0;i<this->matrixSize;i++){
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
00099 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, Markov::API::ModelMatrix::ready, Markov::API::ModelMatrix::totalEdgeWeights, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().

Here is the caller graph for this function:



8.10.3.40 DeallocateMatrix() [3/3]

`bool Markov::API::ModelMatrix::DeallocateMatrix ( ) [protected], [inherited]`

Deallocate matrix and make it ready for re-construction.

Returns

True if deallocated. False if matrix was not initialized

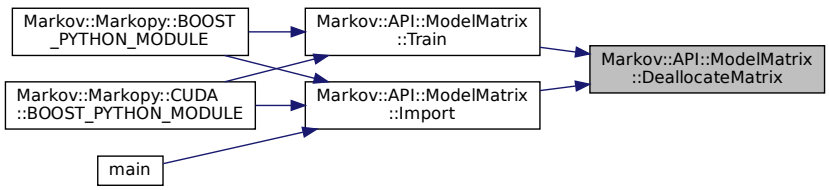
Definition at line 81 of file modelMatrix.cpp.

```
00081                                           {
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
00085 
00086     for(int i=0;i<this->matrixSize;i++){
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090 
00091     for(int i=0;i<this->matrixSize;i++){
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095 
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
00099 }
```

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

Referenced by Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().

Here is the caller graph for this function:



8.10.3.41 DumpJSON() [1/3]

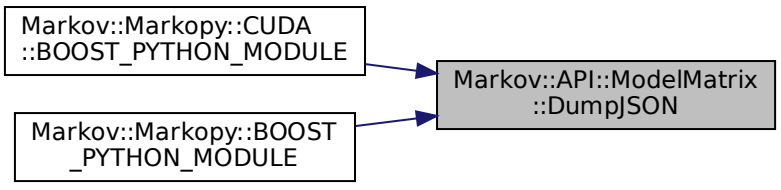
`void Markov::API::ModelMatrix::DumpJSON ( ) [inherited]`

Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.
Definition at line 101 of file modelMatrix.cpp.

```
00101                                {
00102
00103     std::cout « "{\n   \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout « "\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout « "\\\\";
00107         else if(this->matrixIndex[i]==0) std::cout « "\\\\x00";
00108         else if(i==0) std::cout « "\\\\xff";
00109         else if(this->matrixIndex[i]=='\n') std::cout « "\\n";
00110         else std::cout « this->matrixIndex[i];
00111     }
00112     std::cout «
00113     "\",\n"
00114     "   \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++){
00117         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00120         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00121         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00122         for(int j=0;j<this->matrixSize;j++){
00123             if(this->edgeMatrix[i][j]=='"') std::cout « "\"\\\"\"";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout « "\"\\\\\"";
00125             else if(this->edgeMatrix[i][j]==0) std::cout « "\"\\\\x00\"";
00126             else if(this->edgeMatrix[i][j]<0) std::cout « "\"\\\\xff\"";
00127             else if(this->matrixIndex[i]=='\n') std::cout « "\"\\n\"";
00128             else std::cout « "\"" « this->edgeMatrix[i][j] « "\"";
00129             if(j!=this->matrixSize-1) std::cout « ", ";
00130         }
00131         std::cout « "],\n";
00132     }
00133     std::cout « "},\n";
00134
00135     std::cout « "\"   weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++){
00137         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00140         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00141         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++){
00144             std::cout « this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout « ", ";
00146         }
00147         std::cout « "],\n";
00148     }
00149     std::cout « "  }\n}\n";
00150 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), and Markov::Markopy::BOOST_PYTHON_MODULE().
Here is the caller graph for this function:



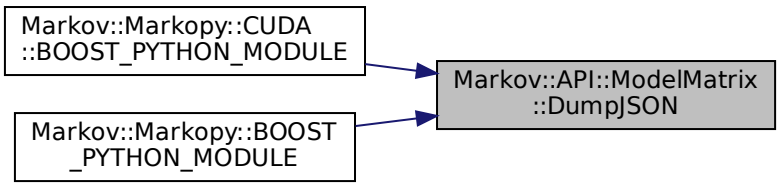
8.10.3.42 DumpJSON() [2/3]

`void Markov::API::ModelMatrix::DumpJSON ( ) [inherited]`
Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.
Definition at line 101 of file modelMatrix.cpp.

```
00101                                {
00102
00103     std::cout « "{\n   \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout « "\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout « "\\\\";
00107         else if(this->matrixIndex[i]==0) std::cout « "\\\\x00";
00108         else if(i==0) std::cout « "\\\\xff";
00109         else if(this->matrixIndex[i]=='\n') std::cout « "\\n";
00110         else std::cout « this->matrixIndex[i];
00111     }
00112     std::cout «
00113     "\",\n"
00114     "   \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++){
00117         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00120         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00121         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00122         for(int j=0;j<this->matrixSize;j++){
00123             if(this->edgeMatrix[i][j]=='"') std::cout « "\"\\\"\"";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout « "\"\\\\\"";
00125             else if(this->edgeMatrix[i][j]==0) std::cout « "\"\\\\x00\"";
00126             else if(this->edgeMatrix[i][j]<0) std::cout « "\"\\\\xff\"";
00127             else if(this->matrixIndex[i]=='\n') std::cout « "\"\\n\"";
00128             else std::cout « "\"" « this->edgeMatrix[i][j] « "\"";
00129             if(j!=this->matrixSize-1) std::cout « ", ";
00130         }
00131         std::cout « "],\n";
00132     }
00133     std::cout « "},\n";
00134
00135     std::cout « "\"   weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++){
00137         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00140         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00141         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++){
00144             std::cout « this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout « ", ";
00146         }
00147         std::cout « "],\n";
00148     }
00149     std::cout « "  }\n}\n";
00150 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), and Markov::Markopy::BOOST_PYTHON_MODULE().
Here is the caller graph for this function:



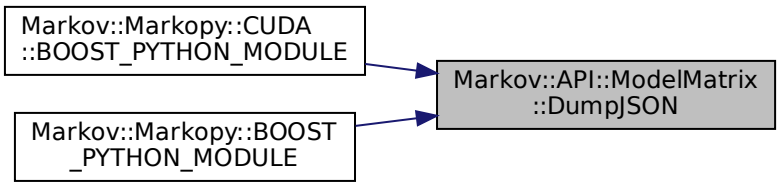
8.10.3.43 DumpJSON() [3/3]

`void Markov::API::ModelMatrix::DumpJSON ( ) [inherited]`
Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.
Definition at line 101 of file modelMatrix.cpp.

```
00101                                {
00102
00103     std::cout « "{\n   \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout « "\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout « "\\\\";
00107         else if(this->matrixIndex[i]==0) std::cout « "\\\\x00";
00108         else if(i==0) std::cout « "\\\\xff";
00109         else if(this->matrixIndex[i]=='\n') std::cout « "\\n";
00110         else std::cout « this->matrixIndex[i];
00111     }
00112     std::cout «
00113     "\",\n"
00114     "   \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++){
00117         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00120         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00121         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00122         for(int j=0;j<this->matrixSize;j++){
00123             if(this->edgeMatrix[i][j]=='"') std::cout « "\"\\\"\"";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout « "\"\\\\\"";
00125             else if(this->edgeMatrix[i][j]==0) std::cout « "\"\\\\x00\"";
00126             else if(this->edgeMatrix[i][j]<0) std::cout « "\"\\\\xff\"";
00127             else if(this->matrixIndex[i]=='\n') std::cout « "\"\\n\"";
00128             else std::cout « "\"" « this->edgeMatrix[i][j] « "\"";
00129             if(j!=this->matrixSize-1) std::cout « ", ";
00130         }
00131         std::cout « "],\n";
00132     }
00133     std::cout « "},\n";
00134
00135     std::cout « "\"   weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++){
00137         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00140         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00141         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++){
00144             std::cout « this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout « ", ";
00146         }
00147         std::cout « "],\n";
00148     }
00149     std::cout « "  }\n}\n";
00150 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), and Markov::Markopy::BOOST_PYTHON_MODULE().
Here is the caller graph for this function:



8.10.3.44 Edges() [1/4]

`std::vector<Edge<char >*>* Markov::Model< char >::Edges ( )  [inline], [inherited]`
Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.10.3.45 Edges() [2/4]

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

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.10.3.46 Edges() [3/4]

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

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.10.3.47 Edges() [4/4]

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

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.10.3.48 evaluate()

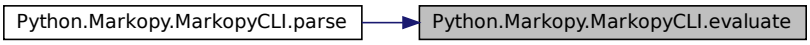
```
def Python.Markopy.MarkopyCLI.evaluate (
              self,
             str filename )  [inherited]
```

Definition at line 163 of file markopy.py.

```
00163     def evaluate(self,filename : str):
00164         if(not self.args.evaluate_type):
00165             if(filename.endswith(".mdl")):
00166                 ModelEvaluator(filename).evaluate()
00167             else:
00168                 CorpusEvaluator(filename).evaluate()
00169         else:
00170             if(self.args.evaluate_type == "model"):
00171                 ModelEvaluator(filename).evaluate()
00172             else:
00173                 CorpusEvaluator(filename).evaluate()
00174
```

Referenced by Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.10.3.49 Export() [1/9]

```
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 166 of file model.h.

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

8.10.3.50 Export() [2/9]

```
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 166 of file model.h.

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

8.10.3.51 Export() [3/9]

```
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 166 of file model.h.

```
00300                                                    {
```

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

8.10.3.52 Export() [4/9]

```
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 166 of file model.h.

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

8.10.3.53 export() [1/6]

```
def Python.Markopy.BaseCLI.export (
             self,
            str filename )  [inherited]
```

Export model to a file.

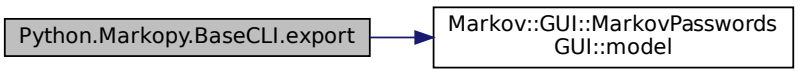
Parameters

filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.54 export() [2/6]

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

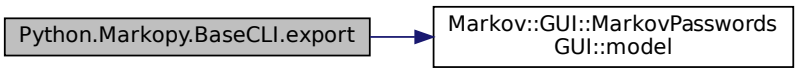
Parameters

filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.55 export() [3/6]

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

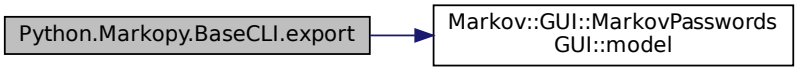
Parameters

filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.56 export() [4/6]

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

Parameters

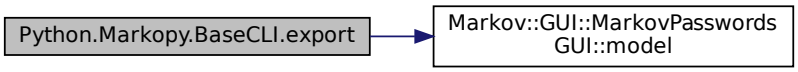
filename	filename to export to

Definition at line 138 of file base.py.

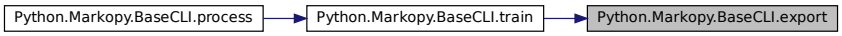
```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.57 export() [5/6]

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

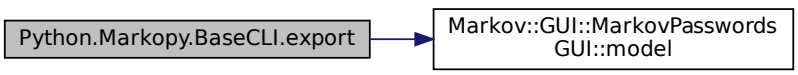
Parameters

filename	filename to export to

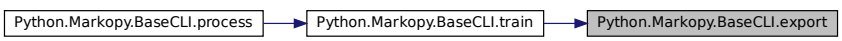
Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.58 export() [6/6]

```
def Python.Markopy.BaseCLI.export (
             self,
            str filename )  [inherited]
```

Export model to a file.

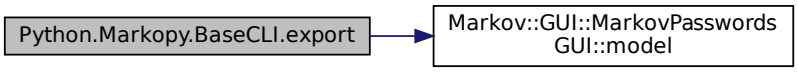
Parameters

filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.59 Export() [5/9]

```
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 155 of file model.h.

```
00288                                                 {
```

```
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
      e->RightNode()->NodeValue() << "\n";
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.10.3.60 Export() [6/9]

```
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 155 of file model.h.

```
00288                                                 {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
      e->RightNode()->NodeValue() << "\n";
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.10.3.61 Export() [7/9]

```
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 155 of file model.h.

```
00288                                                 {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
      e->RightNode()->NodeValue() << "\n";
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.10.3.62 Export() [8/9]

```
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 155 of file model.h.

```
00288                                                   {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
      e->RightNode()->NodeValue() « "\n";
00293         *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.10.3.63 Export() [9/9]

```
def Python.Markopy.MarkovModel.Export (
            str filename )  [inherited]
```

Definition at line 26 of file mm.py.

```
00026     def Export(filename : str):
00027         pass
00028
```

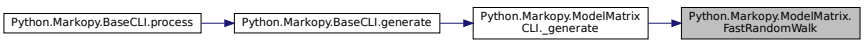
8.10.3.64 FastRandomWalk() [1/9]

```
def Python.Markopy.ModelMatrix.FastRandomWalk (
            int count,
            str wordlist,
            int minlen,
            int maxlen )  [inherited]
```

Definition at line 48 of file mm.py.

```
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass
```

Referenced by Python.Markopy.ModelMatrixCLI._generate().
Here is the caller graph for this function:



8.10.3.65 FastRandomWalk() [2/9]

```
def Python.Markopy.ModelMatrix.FastRandomWalk (
            int count,
            str wordlist,
```

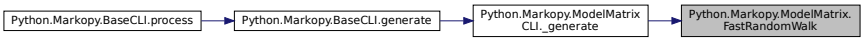
```
            int minlen,
            int maxlen )  [inherited]
```

Definition at line 48 of file mm.py.

```
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass
```

Referenced by Python.Markopy.ModelMatrixCLI._generate().

Here is the caller graph for this function:



8.10.3.66 FastRandomWalk() [3/9]

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

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

TODO

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

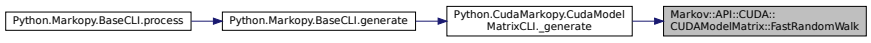
```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 58 of file cudaModelMatrix.cu.

```
00058                                                                                {
00059         cudaDeviceProp prop;
00060         int device=0;
00061         cudaGetDeviceProperties(&prop, device);
00062         cudaChooseDevice(&device, &prop);
00063         //std::cout « "Flattening matrix." « std::endl;
00064         this->FlattenMatrix();
00065         //std::cout « "Migrating matrix." « std::endl;
00066         this->MigrateMatrix();
00067         //std::cout « "Migrated matrix." « std::endl;
00068         std::ofstream wordlist;
00069         if(bFileIO)
00070             wordlist.open(wordlistFileName);
00071
00072
00073         cudaBlocks = 1024;
00074         cudaThreads = 256;
00075         iterationsPerKernelThread = 100;
00076         alternatingKernels = 2;
00077         totalOutputPerKernel = (long int)cudaBlocks*(long int)cudaThreads*iterationsPerKernelThread;
00078         totalOutputPerSync= totalOutputPerKernel*alternatingKernels;
00079         numberOfPartitions = n/totalOutputPerSync;
00080         cudaGridSize = cudaBlocks*cudaThreads;
00081         cudaMemPerGrid = (maxLen+2)*iterationsPerKernelThread;
00082         cudaPerKernelAllocationSize = cudaGridSize*cudaMemPerGrid;
00083         this->prepKernelMemoryChannel(alternatingKernels);
00084
```

```
00085         unsigned long int leftover = n - (totalOutputPerSync*numberOfPartitions);
00086
00087         if(bInfinite && !numberOfPartitions) numberOfPartitions=5;
00088         std::cerr << cudaPerKernelAllocationSize << "\n";
00089
00090         if(n%totalOutputPerSync) std::cerr << "For optimization, request outputs muliples of "<<
      totalOutputPerSync << ".\n";
00091
00092         //start kernelID 1
00093         this->LaunchAsyncKernel(1, minLen, maxLen);
00094
00095         for(int i=1;i<numberOfPartitions;i++){
00096             if(bInfinite) i=0;
00097
00098             //wait kernelID1 to finish, and start kernelID 0
00099             cudaStreamSynchronize(this->cudastreams[1]);
00100             this->LaunchAsyncKernel(0, minLen, maxLen);
00101
00102             //start memcpy from kernel 1 (block until done)
00103             this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00104
00105             //wait kernelID 0 to finish, then start kernelID1
00106             cudaStreamSynchronize(this->cudastreams[0]);
00107             this->LaunchAsyncKernel(1, minLen, maxLen);
00108
00109             //start memcpy from kernel 0 (block until done)
00110             this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00111
00112         }
00113
00114         //wait kernelID1 to finish, and start kernelID 0
00115         cudaStreamSynchronize(this->cudastreams[1]);
00116         this->LaunchAsyncKernel(0, minLen, maxLen);
00117         this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00118         cudaStreamSynchronize(this->cudastreams[0]);
00119         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00120
00121
00122         if(!leftover) return;
00123         alternatingKernels=1;
00124         std::cerr << "Remaining line count (" << leftover << ") is lower than partition. Adjusting CUDA
      workload..\n";
00125         this->iterationsPerKernelThread = leftover/cudaGridSize;
00126         this->LaunchAsyncKernel(0, minLen, maxLen);
00127         cudaStreamSynchronize(this->cudastreams[0]);
00128         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00129
00130         leftover -= this->iterationsPerKernelThread*cudaGridSize;
00131         if(!leftover) return;
00132
00133         std::cerr << "Remaining line count (" << leftover << ") is lower than minimum possible. Handing
      over to CPU generation.\n";
00134         this->iterationsPerKernelThread = leftover/cudaGridSize;
00135
00136         leftover -= this->iterationsPerKernelThread;
00137
00138         if(!leftover) return;
00139         std::cerr << "Remaining " << leftover << " lines are absolutely not worth printing.\n";
00140         Markov::API::ModelMatrix::ConstructMatrix();
00141         Markov::API::ModelMatrix::FastRandomWalk(leftover, &wordlist, minLen, maxLen, 1, bFileIO);
00142
00143     }
```

References Markov::API::CUDA::CUDAModelMatrix::alternatingKernels, Markov::API::CUDA::CUDAModelMatrix::cudaBlocks, Markov::API::CUDA::CUDAModelMatrix::cudaGridSize, Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid, Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize, Markov::API::CUDA::CUDAModelMatrix::cudaThreads, Markov::API::CUDA::CUDAModelMatrix::iterationsPerKernelThread, Markov::API::CUDA::CUDAModelMatrix::numberOfPartitions, Markov::API::CUDA::CUDAModelMatrix::totalOutputPerKernel, and Markov::API::CUDA::CUDAModelMatrix::totalOutputPerSync.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI::_generate().
Here is the caller graph for this function:



8.10.3.67 FastRandomWalk() [4/9]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [inherited]
```

Random walk on the Matrix-reduced Markov::Model.
This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.
While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

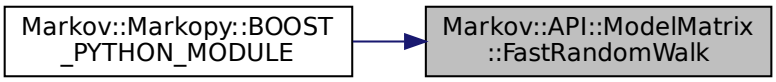
```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 217 of file modelMatrix.cpp.

```
00217                                                {
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

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

Here is the caller graph for this function:



8.10.3.68 FastRandomWalk() [5/9]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [inherited]
```

Random walk on the Matrix-reduced Markov::Model.
This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.
While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

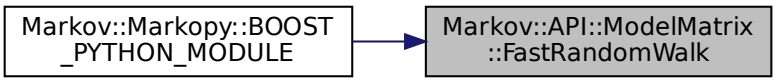
```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 217 of file modelMatrix.cpp.

```
00217
                                                {
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

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

Here is the caller graph for this function:



8.10.3.69 FastRandomWalk() [6/9]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [inherited]
```

Random walk on the Matrix-reduced Markov::Model.
This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.
While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

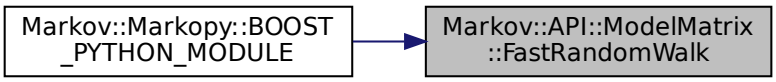
```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 217 of file modelMatrix.cpp.

```
00217                                                {
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

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

Here is the caller graph for this function:



8.10.3.70 FastRandomWalk() [7/9]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [protected], [inherited]
```

Random walk on the Matrix-reduced Markov::Model.
This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.
While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

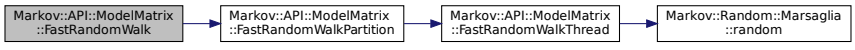
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

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

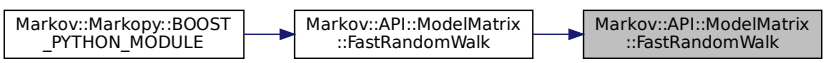
Definition at line 204 of file modelMatrix.cpp.

```
00204
                                    {
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
      threads);
00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
      threads);
00213     }
00214     return 0;
00215 }
```

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



Here is the caller graph for this function:



8.10.3.71 FastRandomWalk() [8/9]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [protected], [inherited]
```

Random walk on the Matrix-reduced Markov::Model.
This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.
While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

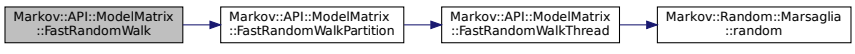
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

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

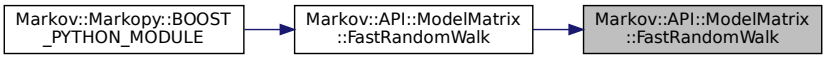
Definition at line 204 of file modelMatrix.cpp.

```
00204
                                    {
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
      threads);
00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
      threads);
00213     }
00214     return 0;
00215 }
```

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



Here is the caller graph for this function:



8.10.3.72 FastRandomWalk() [9/9]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [protected], [inherited]
```

Random walk on the Matrix-reduced Markov::Model.
This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.
While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

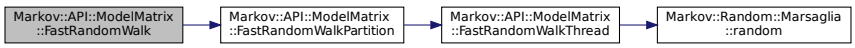
```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 204 of file modelMatrix.cpp.

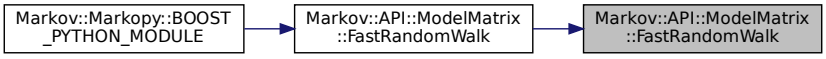
```
00204
                                    {
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
      threads);
00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
      threads);
00213     }
00214     return 0;
00215 }
```

References Markov::API::ModelMatrix::FastRandomWalkPartition().
Referenced by Markov::API::ModelMatrix::FastRandomWalk().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.73 FastRandomWalkPartition() [1/3]

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

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

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

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

Parameters

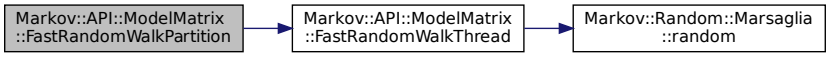
mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads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

Definition at line 225 of file modelMatrix.cpp.

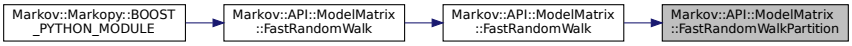
```
00225                                                                    {
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
```

```
00232     int id = 0;
00233     for(int i=0;i<threads;i++){
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
      mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
      wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00239
00240     for(int i=0;i<threads;i++){
00241         threadsV[i]->join();
00242     }
00243 }
```

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



Here is the caller graph for this function:



8.10.3.74 FastRandomWalkPartition() [2/3]

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

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

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

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

Parameters

mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate

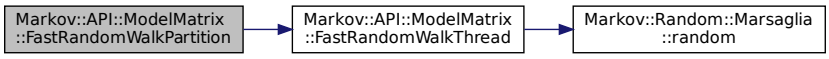
Parameters

maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

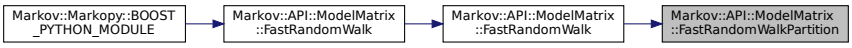
Definition at line 225 of file modelMatrix.cpp.

```
00225                                                                    {
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++){
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
      mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
      wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00239
00240     for(int i=0;i<threads;i++){
00241         threadsV[i]->join();
00242     }
00243 }
```

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



Here is the caller graph for this function:



8.10.3.75 FastRandomWalkPartition() [3/3]

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

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

- 50M results 12 characters max -> 550 Mb Memory allocation

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

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

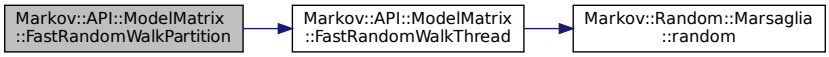
Parameters

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

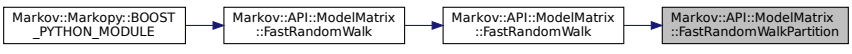
Definition at line 225 of file modelMatrix.cpp.

```
00225                                                                    {
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++){
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
      mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
      wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00239
00240     for(int i=0;i<threads;i++){
00241         threadsV[i]->join();
00242     }
00243 }
```

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



Here is the caller graph for this function:



8.10.3.76 FastRandomWalkThread() [1/3]

```
void Markov::API::ModelMatrix::FastRandomWalkThread (
            std::mutex * mlock,
```

```
            std::ofstream * wordlist,
            unsigned long int n,
            int minLen,
            int maxLen,
            int id,
            bool bFileIO )  [protected], [inherited]
```

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

Parameters

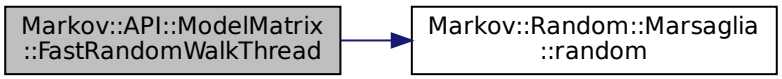
mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
id	- **DEPRECATED** Thread id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

Definition at line 153 of file modelMatrix.cpp.

```
00153                                                                   {
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179
00180             if (len >= maxLen)  break;
00181             else if ((next < 0) && (len < minLen)) continue;
00182             else if (next < 0) break;
00183             cur = next;
00184             res[bufferctr + len++] = cur;
00185         }
00186         res[bufferctr + len++] = '\n';
00187         bufferctr+=len;
00188
00189     }
00190     if(bFileIO){
00191         mlock->lock();
00192         *wordlist << res;
00193         mlock->unlock();
00194     }else{
00195         mlock->lock();
00196         std::cout << res;
00197         mlock->unlock();
00198     }
00199     delete res;
00200
00201 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, Markov::Random::Marsaglia::random(), Markov::API::ModelMatrix::totalEdgeWeights, and Markov::API::ModelMatrix::valueMatrix.

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



Here is the caller graph for this function:

8.10.3.77 FastRandomWalkThread() [2/3]

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

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

Parameters

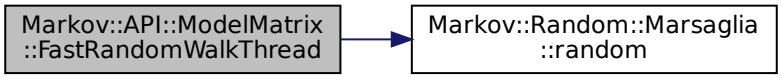
mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
id	- **DEPRECATED** Thread id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

Definition at line 153 of file modelMatrix.cpp.

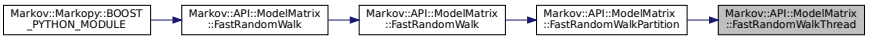
```
00153                                                                {
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
```

```
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179 
00180             if (len >= maxLen)  break;
00181             else if ((next < 0) && (len < minLen)) continue;
00182             else if (next < 0) break;
00183             cur = next;
00184             res[bufferctr + len++] = cur;
00185         }
00186         res[bufferctr + len++] = '\n';
00187         bufferctr+=len;
00188 
00189     }
00190     if(bFileIO){
00191         mlock->lock();
00192         *wordlist << res;
00193         mlock->unlock();
00194     }else{
00195         mlock->lock();
00196         std::cout << res;
00197         mlock->unlock();
00198     }
00199     delete res;
00200 
00201 }
```

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



Here is the caller graph for this function:



8.10.3.78 FastRandomWalkThread() [3/3]

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

A single thread of a single partition of FastRandomWalk.
A FastRandomWalkPartition will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

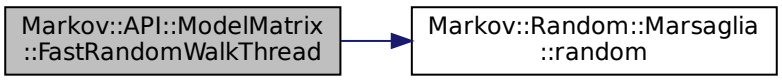
mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
id	- **DEPRECATED** Thread id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

Definition at line 153 of file modelMatrix.cpp.

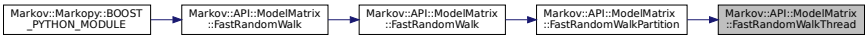
```
00153                                                                {
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179
00180             if (len >= maxLen)  break;
00181             else if ((next < 0) && (len < minLen)) continue;
00182             else if (next < 0) break;
00183             cur = next;
00184             res[bufferctr + len++] = cur;
00185         }
00186         res[bufferctr + len++] = '\n';
00187         bufferctr+=len;
00188
00189     }
00190     if(bFileIO){
00191         mlock->lock();
00192         *wordlist << res;
00193         mlock->unlock();
00194     }else{
00195         mlock->lock();
00196         std::cout << res;
00197         mlock->unlock();
00198     }
00199     delete res;
00200
00201 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, Markov::Random::Marsaglia::random(), Markov::API::ModelMatrix::totalEdgeWeights, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::API::ModelMatrix::FastRandomWalkPartition().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.79 FlattenMatrix()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix ( )  [inherited]
```

Flatten migrated matrix from 2d to 1d.

Definition at line 261 of file cudaModelMatrix.cu.

```
00261                                                       {
00262         this->flatEdgeMatrix = new char[this->matrixSize*this->matrixSize];
00263 
00264         this->flatValueMatrix = new long int[this->matrixSize*this->matrixSize];
00265         for(int i=0;i<this->matrixSize;i++){
00266             memcpy(&this->flatEdgeMatrix[i*this->matrixSize], this->edgeMatrix[i], this->matrixSize );
00267             memcpy(&this->flatValueMatrix[i*this->matrixSize], this->valueMatrix[i],
      this->matrixSize*sizeof(long int) );
00268         }
00269     }
```

References Markov::API::CUDA::CUDAModelMatrix::flatEdgeMatrix, Markov::API::CUDA::CUDAModelMatrix::flatValueMatrix, and Markov::API::ModelMatrix::matrixSize.

8.10.3.80 GatherAsyncKernelOutput()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput (
             int kernelID,
             bool bFileIO,
             std::ofstream & wordlist )  [protected], [inherited]
```

Definition at line 180 of file cudaModelMatrix.cu.

```
00180 
                      {
00181 
      cudaMemcpy(this->outputBuffer[kernelID],this->device_outputBuffer[kernelID],cudaPerKernelAllocationSize,
      cudaMemcpyDeviceToHost);
00182         //std::cerr « "Kernel" « kernelID « " output copied\n";
00183         if(bFileIO){
00184             for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00185                 wordlist « &this->outputBuffer[kernelID][j];
00186             }
00187         }else{
00188             for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00189                 std::cout « &this->outputBuffer[kernelID][j];
00190             }
00191         }
00192     }
```

References Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid, Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAlloc and Markov::API::CUDA::CUDAModelMatrix::outputBuffer.

8.10.3.81 Generate() [1/5]

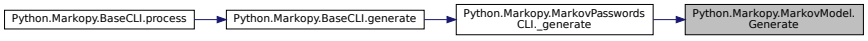
```
def Python.Markopy.MarkovModel.Generate (
            int count,
            str wordlist,
            int minlen,
            int maxlen,
            int threads )  [inherited]
```

Definition at line 34 of file mm.py.

```
00034     def Generate(count : int, wordlist : str, minlen : int, maxlen: int, threads : int):
00035         pass
00036
00037
```

Referenced by Python.Markopy.MarkovPasswordsCLI._generate().

Here is the caller graph for this function:



8.10.3.82 generate() [1/6]

```
def Python.Markopy.BaseCLI.generate (
             self,
            str wordlist,
            bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

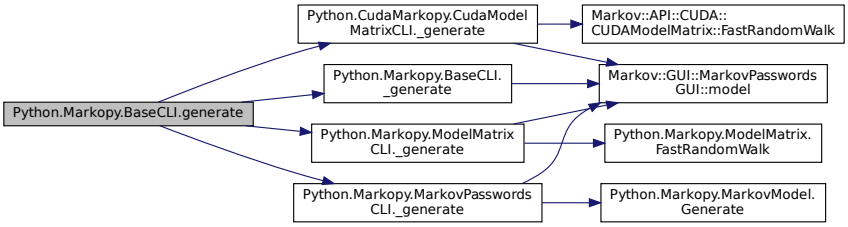
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.83 generate() [2/6]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

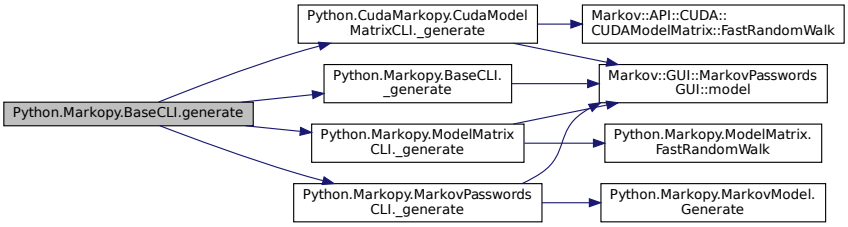
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159 
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.84 generate() [3/6]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

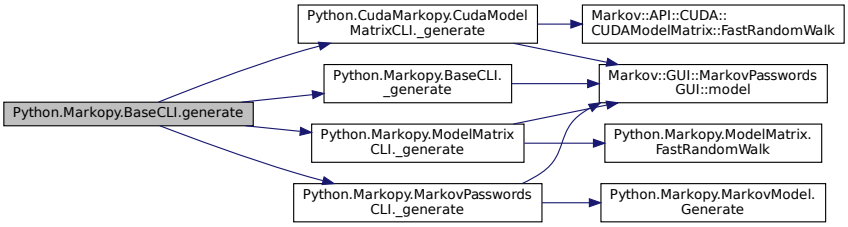
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159 
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.85 generate() [4/6]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

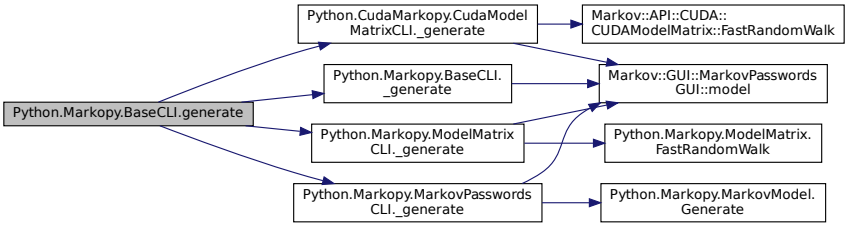
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159 
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.86 generate() [5/6]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

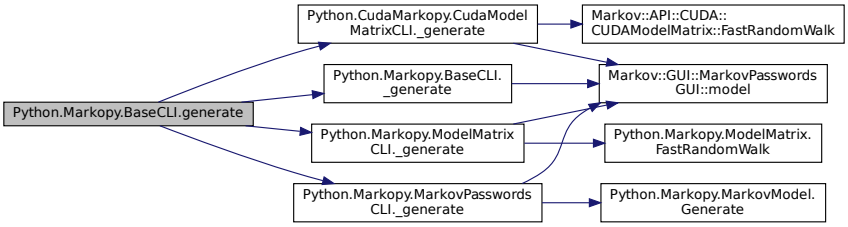
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159 
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.87 generate() [6/6]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

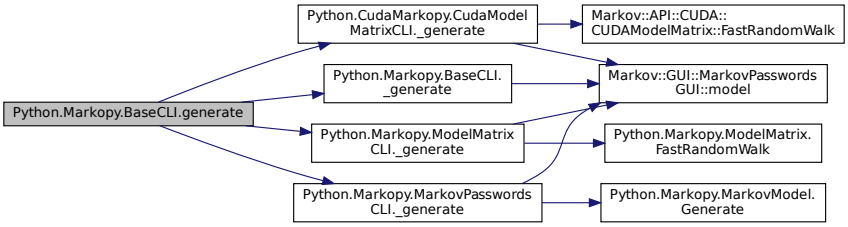
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159 
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.88 Generate() [2/5]

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

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

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

Parameters

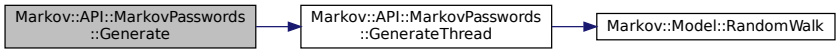
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

Definition at line 118 of file markovPasswords.cpp.

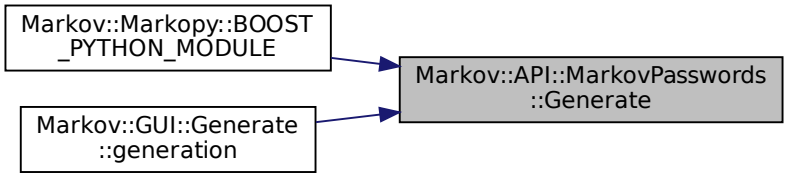
```
00118                                {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
       &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
```

```
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

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



Here is the caller graph for this function:



8.10.3.89 Generate() [3/5]

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

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

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

Parameters

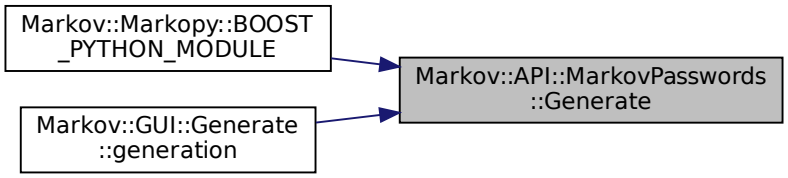
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

Definition at line 118 of file markovPasswords.cpp.

```
00118
                                {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
      &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

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

Here is the caller graph for this function:



8.10.3.90 Generate() [4/5]

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

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

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

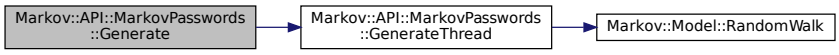
Parameters

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

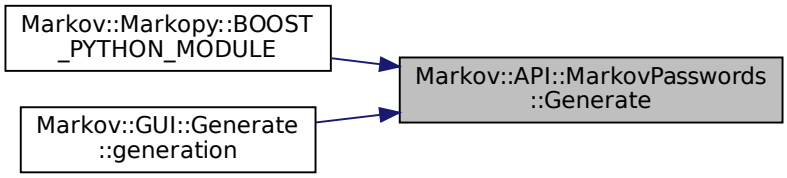
Definition at line 118 of file markovPasswords.cpp.

```
00118
                            {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
      &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

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



Here is the caller graph for this function:



8.10.3.91 Generate() [5/5]

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

```
            int maxLen = 12,
            int threads = 20 )  [inherited]
```

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

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

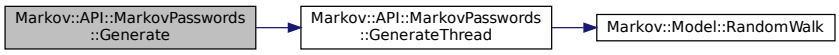
Parameters

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

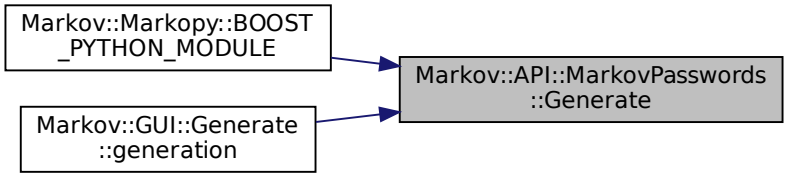
Definition at line 118 of file markovPasswords.cpp.

```
00118                                {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
      &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

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



Here is the caller graph for this function:



8.10.3.92 GenerateThread()

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

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

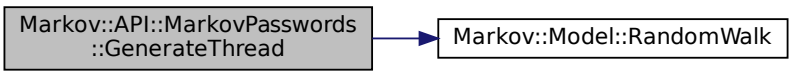
Parameters

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

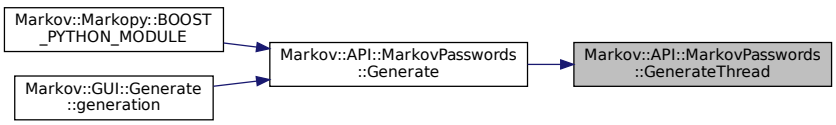
Definition at line 140 of file markovPasswords.cpp.

```
00140
                                    {
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist « res « "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

References Markov::Model< NodeStorageType >::RandomWalk().
Referenced by Markov::API::MarkovPasswords::Generate().
Here is the call graph for this function:



Here is the caller graph for this function:



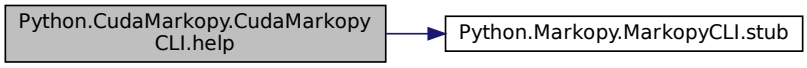
8.10.3.93 help()

```
def Python.CudaMarkopy.CudaMarkopyCLI.help (
              self )
```

overload help function to print submodel helps
Reimplemented from Python.Markopy.MarkopyCLI.
Definition at line 61 of file cudamarkopy.py.

```
00061     def help(self):
00062         "! @brief overload help string"
00063         self.parser.print_help = self.stub
00064         self.args = self.parser.parse_known_args()[0]
00065         if(self.args.model_type!="_MMX"):
00066             if(self.args.model_type=="MP"):
00067                 mp = MarkovPasswordsCLI()
00068                 mp.add_arguments()
00069                 mp.parser.print_help()
00070             elif(self.args.model_type=="MMX"):
00071                 mp = ModelMatrixCLI()
00072                 mp.add_arguments()
00073                 mp.parser.print_help()
00074             elif(self.args.model_type == "CUDA"):
00075                 mp = CudaModelMatrixCLI()
00076                 mp.add_arguments()
00077                 mp.parser.print_help()
00078         else:
00079             print(colored("Model Mode selection choices:", "green"))
00080             self.print_help()
00081             print(colored("Following are applicable for -mt MP mode:", "green"))
00082             mp = MarkovPasswordsCLI()
00083             mp.add_arguments()
00084             mp.parser.print_help()
00085             print(colored("Following are applicable for -mt MMX mode:", "green"))
00086             mp = ModelMatrixCLI()
00087             mp.add_arguments()
00088             mp.parser.print_help()
00089             print(colored("Following are applicable for -mt CUDA mode:", "green"))
00090             mp = CudaModelMatrixCLI()
00091             mp.add_arguments()
00092             mp.parser.print_help()
00093         exit()
00094
```

References Python.Markopy.BaseCLI.parser, and Python.Markopy.MarkopyCLI.stub().
Referenced by Python.Markopy.MarkopyCLI.add_arguments().
Here is the call graph for this function:



Here is the caller graph for this function:

8.10.3.94 Import() [1/8]

```
void Markov::API::ModelMatrix::Import (
             const char * filename )  [inherited]
```

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

Returns

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

Example Use: Import a file with filename

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

Construct the matrix when done.

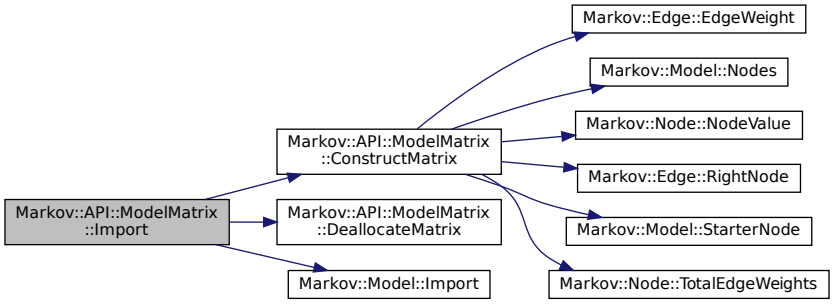
Definition at line 19 of file modelMatrix.cpp.

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

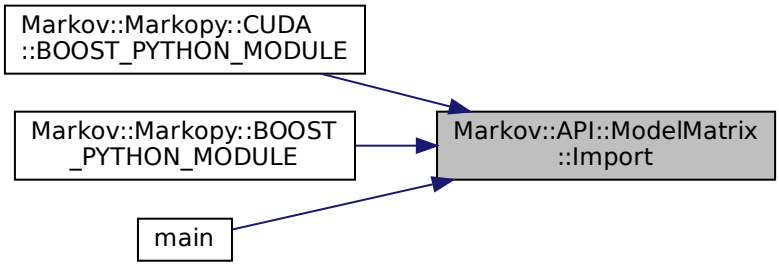
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::Model< NodeStorageType >::Import().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.95 Import() [2/8]

```
void Markov::API::ModelMatrix::Import (
            const char * filename )  [inherited]
```

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

Returns

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

Example Use: Import a file with filename

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

Construct the matrix when done.

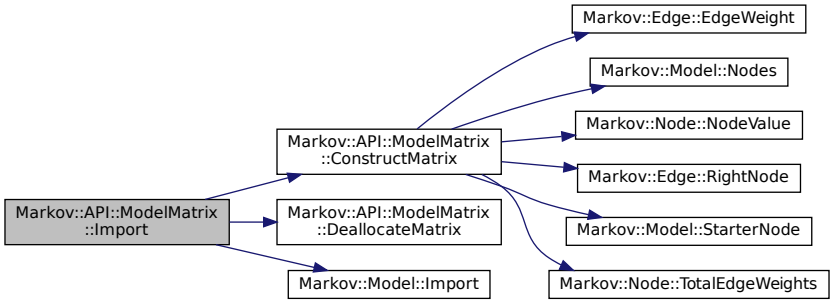
Definition at line 19 of file modelMatrix.cpp.

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

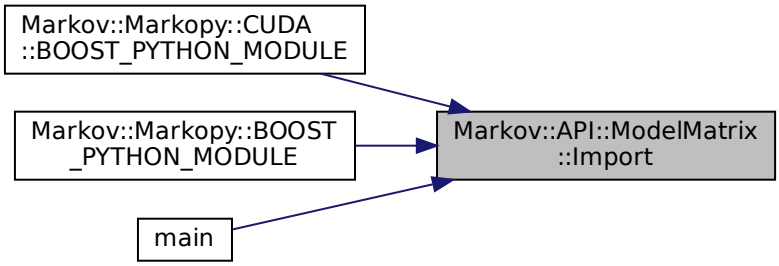
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::Model< NodeStorageType >::Import().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.96 Import() [3/8]

```
void Markov::API::ModelMatrix::Import (
            const char * filename )  [inherited]
```

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

Returns

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

Example Use: Import a file with filename

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

Construct the matrix when done.

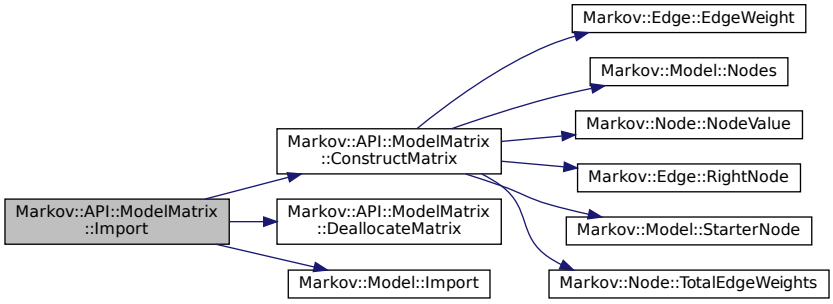
Definition at line 19 of file modelMatrix.cpp.

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

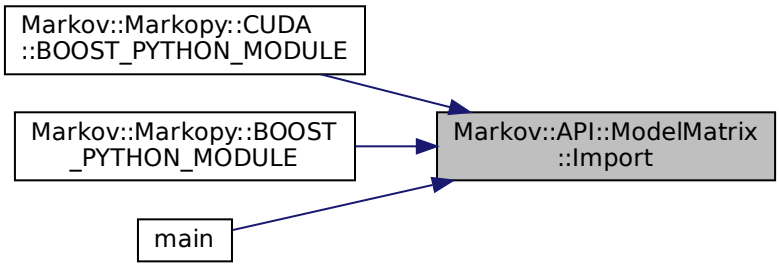
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::Model< NodeStorageType >::Import().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.97 Import() [4/8]

```
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 126 of file model.h.

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

8.10.3.98 Import() [5/8]

```
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 126 of file model.h.

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

```
00219     char src;
00220     char target;
00221     long int oc;
00222 
00223     while (std::getline(*f, cell)) {
00224         //std::cout « "cell: " « cell « std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout « oc « "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout « "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241 
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout « "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253 
00254         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
00255 
00256 
00257     }
00258 
00259     this->OptimizeEdgeOrder();
00260 
00261     return true;
00262 }
```

8.10.3.99 Import() [6/8]

```
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 126 of file model.h.

```
00216                                                   {
00217     std::string cell;
00218 
00219     char src;
00220     char target;
00221     long int oc;
00222 
00223     while (std::getline(*f, cell)) {
00224         //std::cout « "cell: " « cell « std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout « oc « "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
```

```
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout « "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout « "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

8.10.3.100 Import() [7/8]

```
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 126 of file model.h.

```
00216                                                   {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout « "cell: " « cell « std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout « oc « "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout « "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
```

```
00245             //std::cout « "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253 
00254         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
00255 
00256 
00257     }
00258 
00259     this->OptimizeEdgeOrder();
00260 
00261     return true;
00262 }
```

8.10.3.101 Import() [8/8]

```
def Python.Markopy.MarkovModel.Import (
             str filename )  [inherited]
```

Definition at line 22 of file mm.py.

```
00022     def Import(filename : str):
00023         pass
00024 
```

8.10.3.102 import_model() [1/6]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

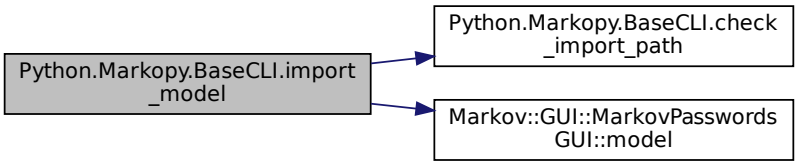
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

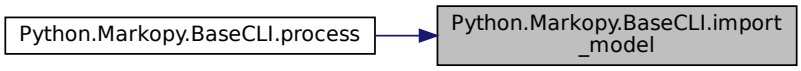
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.103 import_model() [2/6]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

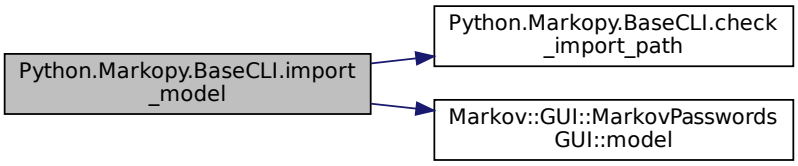
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

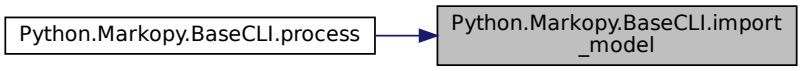
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.104 import_model() [3/6]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

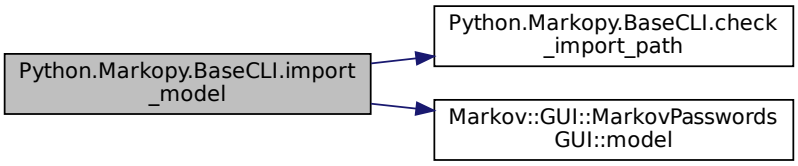
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

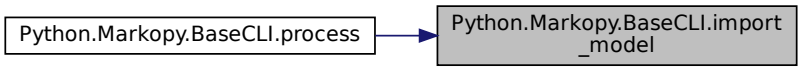
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.105 import_model() [4/6]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

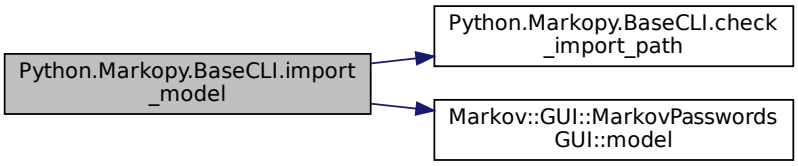
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

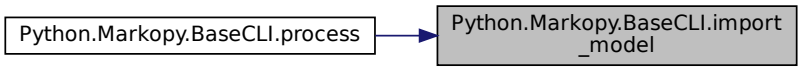
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.106 import_model() [5/6]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

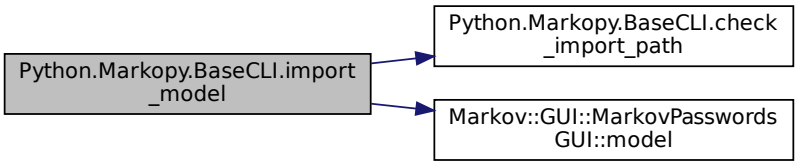
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

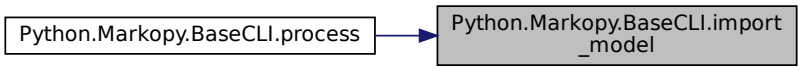
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.107 import_model() [6/6]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

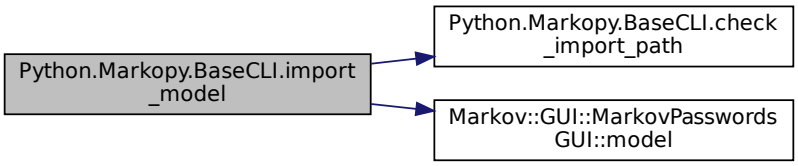
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

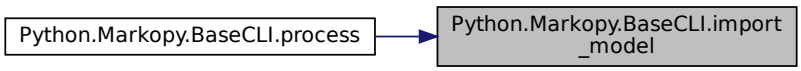
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.108 init_post_arguments() [1/3]

```
def Python.Markopy.MarkopyCLI.init_post_arguments (
              sel ) [inherited]
```

Reimplemented from Python.Markopy.BaseCLI.

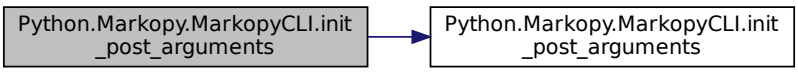
Definition at line 175 of file markopy.py.

```
00175     def init_post_arguments(sel):
00176         pass
00177
```

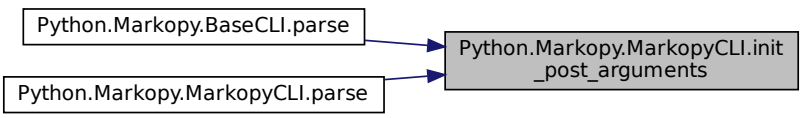
References Python.Markopy.MarkopyCLI.init_post_arguments().

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.109 init_post_arguments() [2/3]

```
def Python.CudaMarkopy.CudaModelMatrixCLI.init_post_arguments (
              self )  [inherited]
```

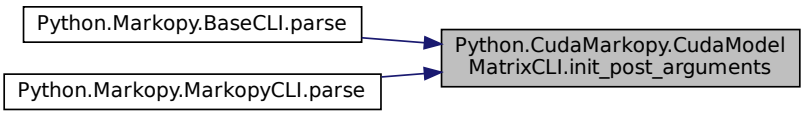
Reimplemented from Python.Markopy.ModelMatrixCLI.

Definition at line 71 of file cudammx.py.

```
00071     def init_post_arguments(self):
00072         super().init_post_arguments()
00073         self.bInfinite = self.args.infinite
00074
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.10.3.110 init_post_arguments() [3/3]

```
def Python.Markopy.MarkopyCLI.init_post_arguments (
              self )  [inherited]
```

Reimplemented from Python.Markopy.BaseCLI.

Definition at line 145 of file markopy.py.

```
00145     def init_post_arguments(self):
00146         pass
00147
```

Referenced by Python.Markopy.MarkopyCLI.init_post_arguments(), Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:

8.10.3.111 LaunchAsyncKernel()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::LaunchAsyncKernel (
             int kernelID,
             int minLen,
             int maxLen )  [protected], [inherited]
```

Definition at line 171 of file cudaModelMatrix.cu.

```
00171                                                                                                                {
00172
00173         //if(kernelID == 0);// cudaStreamSynchronize(this->cudastreams[2]);
00174         //else cudaStreamSynchronize(this->cudastreams[kernelID-1]);
00175         FastRandomWalkCUDAKernel<<<cudaBlocks,cudaThreads,0,
      this->cudastreams[kernelID]>>>(iterationsPerKernelThread, minLen, maxLen,
      this->device_outputBuffer[kernelID], this->device_matrixIndex,
00176             this->device_totalEdgeWeights, this->device_valueMatrix, this->device_edgeMatrix,
      this->matrixSize, cudaMemPerGrid, this->device_seeds[kernelID]);
00177         //std::cerr << "Started kernel" << kernelID << "\n";
00178     }
```

8.10.3.112 ListCudaDevices()

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

List CUDA devices in the system.

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

Example output:

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

Definition at line 16 of file cudaDeviceController.cu.

```
00016                                                       { //list cuda Capable devices on
      host.
00017         int nDevices;
00018         cudaGetDeviceCount(&nDevices);
00019         for (int i = 0; i < nDevices; i++) {
00020             cudaDeviceProp prop;
00021             cudaGetDeviceProperties(&prop, i);
00022             std::cerr « "Device Number: " «  i  « "\n";
00023             std::cerr « "Device name: " « prop.name « "\n";
00024             std::cerr « "Memory Clock Rate (KHz): " « prop.memoryClockRate « "\n";
00025             std::cerr « "Memory Bus Width (bits): " « prop.memoryBusWidth « "\n";
00026             std::cerr « "Peak Memory Bandwidth (GB/s): " « 2.0 * prop.memoryClockRate *
      (prop.memoryBusWidth / 8) / 1.0e6 « "\n";
00027             std::cerr « "Max Linear Threads: " « prop.maxThreadsDim[0] « "\n";
00028
00029         }
00030     }
```

8.10.3.113 MigrateMatrix()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::MigrateMatrix ( )  [inherited]
```

Migrate the class members to the VRAM.

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

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

Definition at line 20 of file cudaModelMatrix.cu.

```
00020                                                   {
00021         cudaError_t cudastatus;
00022
00023         cudastatus = cudaMalloc((char**)&(this->device_matrixIndex),
00024             this->matrixSize*sizeof(char));
00025         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_matrixIndex.");
00026
00027         cudastatus = cudaMalloc((long int **)&(this->device_totalEdgeWeights),
      this->matrixSize*sizeof(long int));
00028         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_totalEdgeWeights.");
00029
00030         cudastatus = cudaMemcpy(this->device_matrixIndex, this->matrixIndex,
00031             this->matrixSize*sizeof(char), cudaMemcpyHostToDevice);
00032         CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Index)");
00033
00034         cudastatus = cudaMemcpy(this->device_totalEdgeWeights, this->totalEdgeWeights,
00035             this->matrixSize*sizeof(long int), cudaMemcpyHostToDevice);
00036         CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Total Edge Values)");
00037
00038         cudastatus = CudaMigrate2DFlat<char>(
00039             &(this->device_edgeMatrix), this->edgeMatrix, this->matrixSize, this->matrixSize);
00040         CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize edge matrix.");
00041
00042         cudastatus = CudaMigrate2DFlat<long int>(
00043             &(this->device_valueMatrix), this->valueMatrix, this->matrixSize, this->matrixSize);
00044         CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
00045
00046     }
```

8.10.3.114 Nodes() [1/4]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.10.3.115 Nodes() [2/4]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.10.3.116 Nodes() [3/4]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.10.3.117 Nodes() [4/4]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.10.3.118 OpenDatasetFile() [1/4]

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

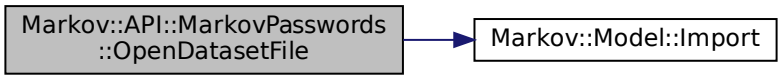
ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
```

```
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References Markov::Model< NodeStorageType >::Import().
Here is the call graph for this function:



8.10.3.119 OpenDatasetFile() [2/4]

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

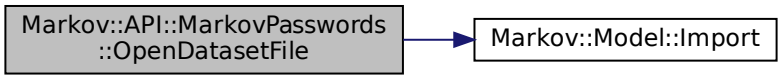
Returns

ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References Markov::Model< NodeStorageType >::Import().
Here is the call graph for this function:



8.10.3.120 OpenDatasetFile() [3/4]

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

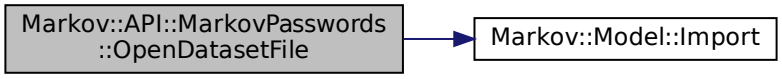
ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

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

Here is the call graph for this function:



8.10.3.121 OpenDatasetFile() [4/4]

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

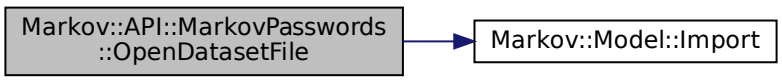
ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

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

Here is the call graph for this function:



8.10.3.122 OptimizeEdgeOrder() [1/4]

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

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.10.3.123 OptimizeEdgeOrder() [2/4]

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

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.10.3.124 OptimizeEdgeOrder() [3/4]

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

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.10.3.125 OptimizeEdgeOrder() [4/4]

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

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.10.3.126 parse()

def Python.CudaMarkopy.CudaMarkopyCLI.parse (
self)

Reimplemented from Python.Markopy.MarkopyCLI.

Definition at line 95 of file cudamarkopy.py.

```
00095     def parse(self):
00096         markopy.MarkopyCLI.parse(self)
00097
```

8.10.3.127 parse_arguments() [1/6]

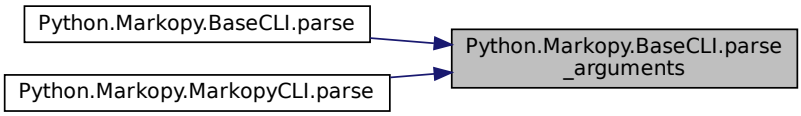
def Python.Markopy.BaseCLI.parse_arguments (
self) [inherited]

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.10.3.128 parse_arguments() [2/6]

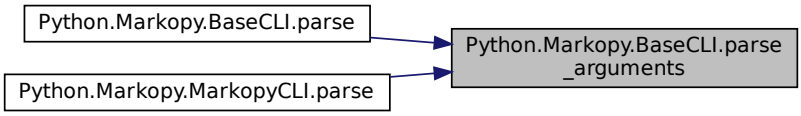
```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.10.3.129 parse_arguments() [3/6]

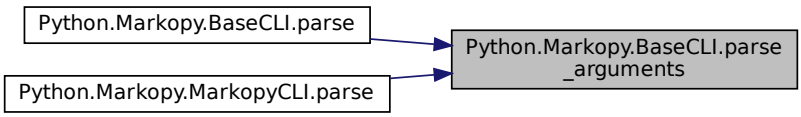
```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.10.3.130 parse_arguments() [4/6]

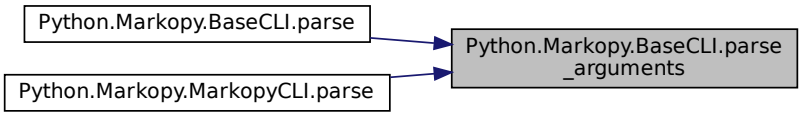
```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.10.3.131 parse_arguments() [5/6]

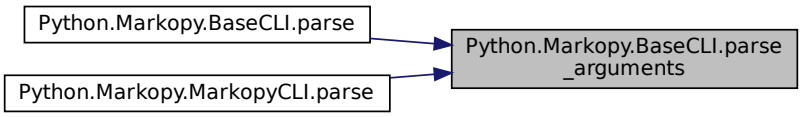
```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.10.3.132 parse_arguments() [6/6]

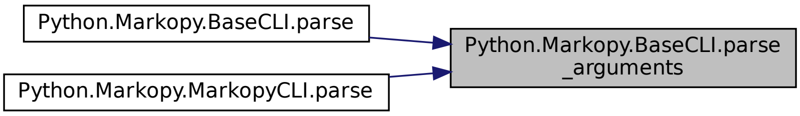
```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.10.3.133 parse_fail()

```
def Python.CudaMarkopy.CudaMarkopyCLI.parse_fail (
                self )
```

Reimplemented from Python.Markopy.MarkopyCLI.

Definition at line 98 of file cudamarkopy.py.

```
    def parse_fail(self):
        "! @brief Not a valid model type"
        if(self.args.model_type == "CUDA"):
            self.cli = CudaModelMatrixCLI()
        else:
            markopy.MarkopyCLI.parse_fail(self)


```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

8.10.3.134 prepKernelMemoryChannel()

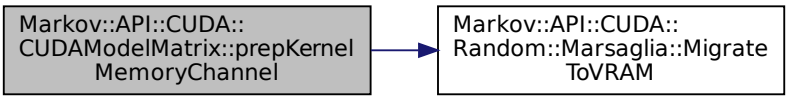
```
__host__ void Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel (
               int numberOfStreams ) [protected], [inherited]
```

Definition at line 145 of file cudaModelMatrix.cu.

```
                                                                                    {

        this->cudastreams = new cudaStream_t[numberOfStreams];
        for(int i=0;i<numberOfStreams;i++)
            cudaStreamCreate(&this->cudastreams[i]);

        this-> outputBuffer = new char*[numberOfStreams];
        for(int i=0;i<numberOfStreams;i++)
            this->outputBuffer[i]= new char[cudaPerKernelAllocationSize];

        cudaError_t cudastatus;
        this-> device_outputBuffer = new char*[numberOfStreams];
            for(int i=0;i<numberOfStreams;i++){
                cudastatus = cudaMalloc((char**)&(device_outputBuffer[i]),
cudaPerKernelAllocationSize);
                CudaCheckNotifyErr(cudastatus, "Failed to establish memory channel. Possibly out of
VRAM?");
            }

        this-> device_seeds = new unsigned long*[numberOfStreams];
        for(int i=0;i<numberOfStreams;i++){
            Markov::API::CUDA::Random::Marsaglia *MEarr = new
Markov::API::CUDA::Random::Marsaglia[cudaGridSize];
            this->device_seeds[i] = Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(MEarr,
cudaGridSize);
            delete[] MEarr;
        }

    }
```

References Markov::API::CUDA::CUDAModelMatrix::cudaGridSize, Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocatio
Markov::API::CUDA::CUDAModelMatrix::device_outputBuffer, Markov::API::CUDA::CUDAModelMatrix::device_seeds,
Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(), and Markov::API::CUDA::CUDAModelMatrix::outputBuffer.

Here is the call graph for this function:



8.10.3.135 process()

```
def Python.Markopy.MarkopyCLI.process (
              self )  [inherited]
```

Process parameters for operation.

Reimplemented from Python.Markopy.BaseCLI.

Definition at line 154 of file markopy.py.

```
00154     def process(self):
00155         "! @brief pass the process request to selected submodel"
00156         return self.cli.process()
00157
```

References Python.CudaMarkopy.CudaMarkopyCLI.cli, and Python.Markopy.MarkopyCLI.cli.

8.10.3.136 RandomWalk() [1/4]

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307                                                       {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319 
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323 
00324         n = temp_node;
00325 
00326         buffer[len++] = n->NodeValue();
00327     }
00328 
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331 
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.10.3.137 RandomWalk() [2/4]

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307                                                           {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319 
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323 
00324         n = temp_node;
00325 
00326         buffer[len++] = n->NodeValue();
00327     }
00328 
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331 
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.10.3.138 RandomWalk() [3/4]

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307                                                       {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319 
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323 
00324         n = temp_node;
00325 
00326         buffer[len++] = n->NodeValue();
00327     }
00328 
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331 
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.10.3.139 RandomWalk() [4/4]

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307                                                       {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319 
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323 
00324         n = temp_node;
00325 
00326         buffer[len++] = n->NodeValue();
00327     }
00328 
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331 
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.10.3.140 Save() [1/4]

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

Export model to file.

Parameters

filename	- Export filename.

Returns

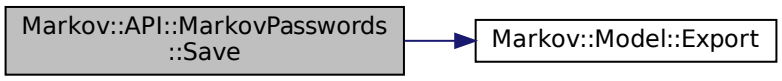
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                   {
00107     std::ofstream* exportFile;
00108 
00109     std::ofstream newFile(filename);
00110 
00111     exportFile = &newFile;
00112 
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

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

Here is the call graph for this function:



8.10.3.141 Save() [2/4]

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

Export model to file.

Parameters

filename	- Export filename.

Returns

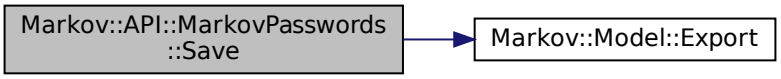
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                    {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

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

Here is the call graph for this function:



8.10.3.142 Save() [3/4]

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

Export model to file.

Parameters

filename	- Export filename.

Returns

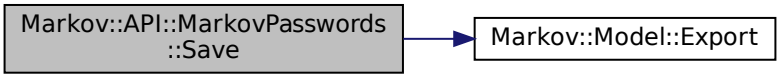
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                    {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
```

```
00115 }
```

References Markov::Model< NodeStorageType >::Export().
Here is the call graph for this function:



8.10.3.143 Save() [4/4]

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

Export model to file.

Parameters

filename	- Export filename.

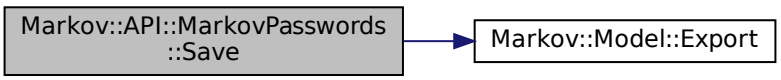
Returns

std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References Markov::Model< NodeStorageType >::Export().
Here is the call graph for this function:



8.10.3.144 StarterNode() [1/4]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.10.3.145 StarterNode() [2/4]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.10.3.146 StarterNode() [3/4]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.10.3.147 StarterNode() [4/4]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.10.3.148 stub()

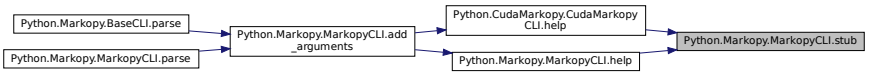
```
def Python.Markopy.MarkopyCLI.stub (
              self )  [inherited]
```

Definition at line 158 of file markopy.py.

```
00158     def stub(self):
00159         "! @brief stub function to hack help requests"
00160         return
00161
00162
```

Referenced by Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.help().

Here is the caller graph for this function:



8.10.3.149 Train() [1/4]

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

Train the model with the dataset file.

Parameters

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

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

Construct the matrix when done.

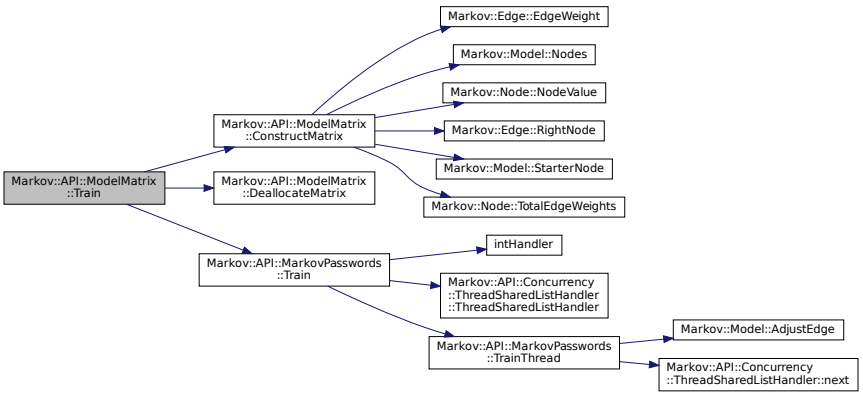
Definition at line 25 of file modelMatrix.cpp.

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

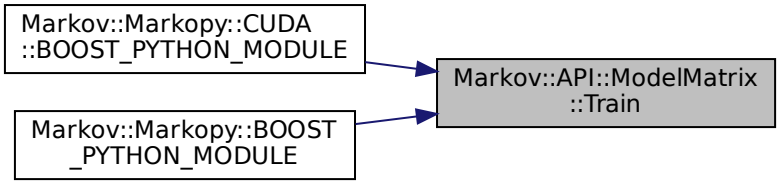
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::MarkovPasswords::Train().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.150 Train() [2/4]

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

Train the model with the dataset file.

Parameters

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

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

Construct the matrix when done.

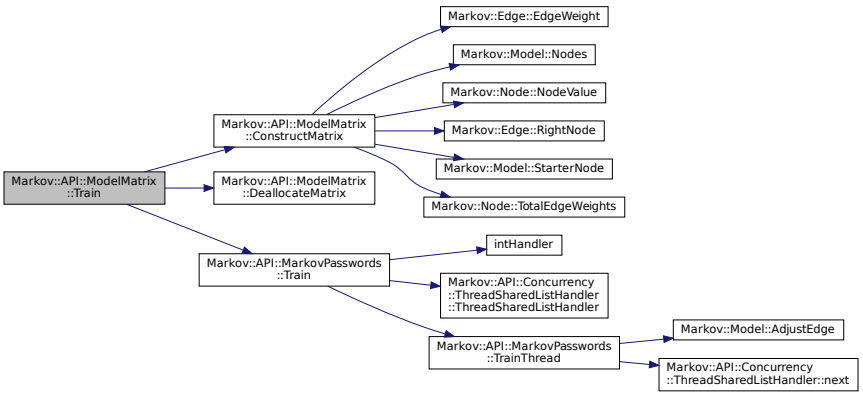
Definition at line 25 of file modelMatrix.cpp.

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

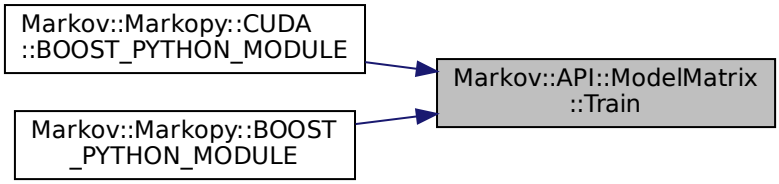
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::MarkovPasswords::Train().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.151 Train() [3/4]

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

Train the model with the dataset file.

Parameters

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

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

Construct the matrix when done.

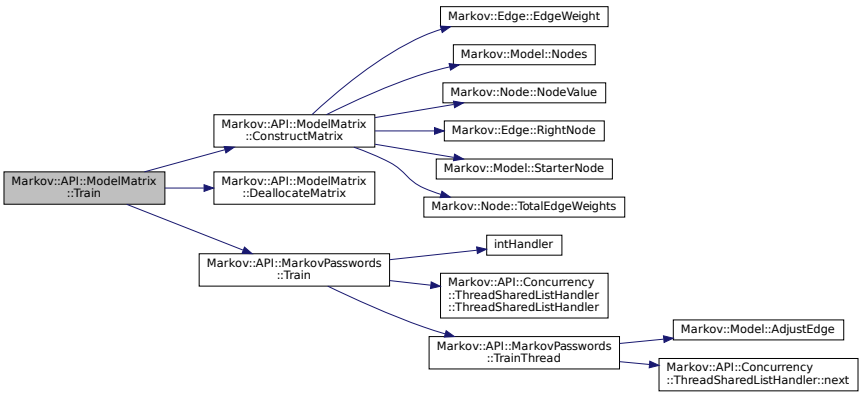
Definition at line 25 of file modelMatrix.cpp.

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

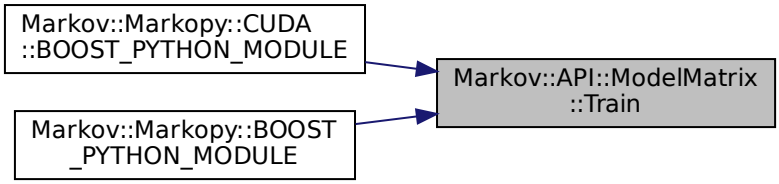
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::MarkovPasswords::Train().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.152 train() [1/6]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

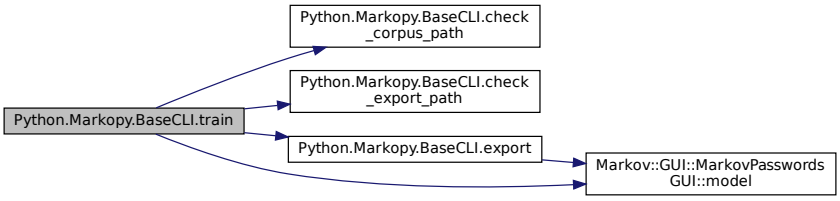
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

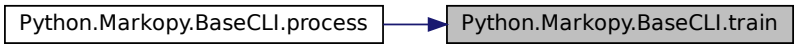
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
       bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
       else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
       accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.153 train() [2/6]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

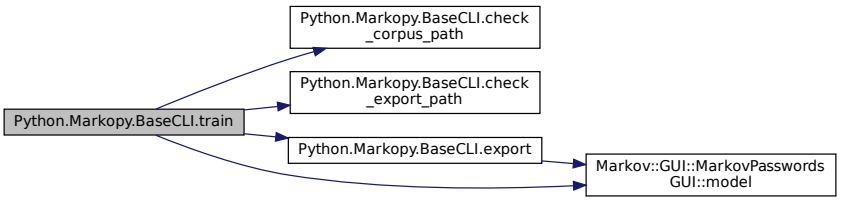
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

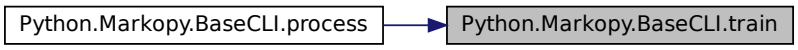
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
```

```
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.154 train() [3/6]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

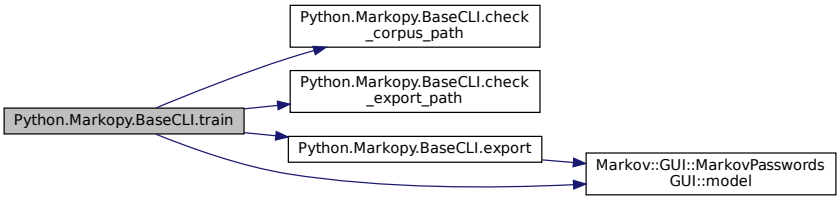
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

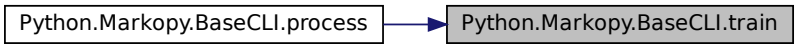
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.155 train() [4/6]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

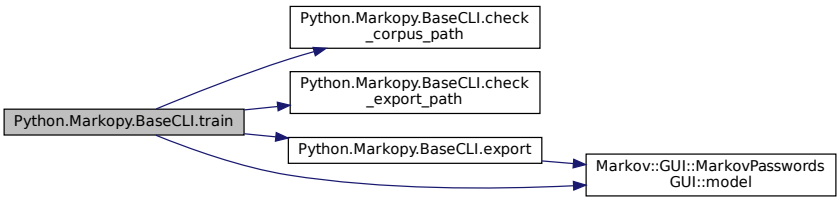
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

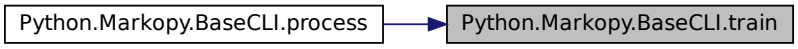
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
       bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
       else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
```

```
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.156 train() [5/6]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

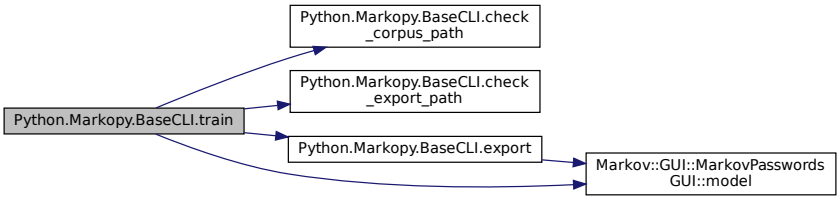
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

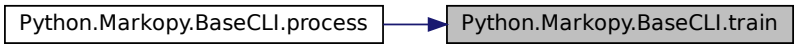
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else''} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.157 train() [6/6]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

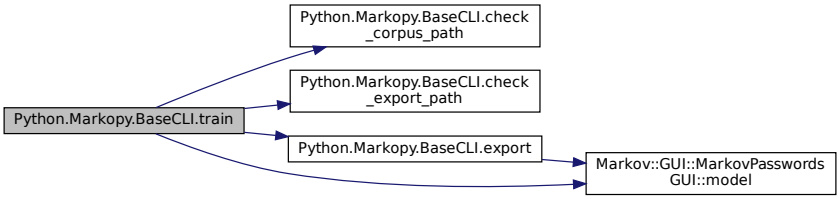
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

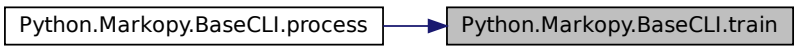
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
       bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
       else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
```

```
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.158 Train() [4/4]

```
def Python.Markopy.MarkovModel.Train (
            str dataset,
            str seperator,
            int threads )  [inherited]
```

Definition at line 30 of file mm.py.

```
00030     def Train(dataset: str, seperator : str, threads : int):
00031         pass
00032 
```

8.10.3.159 TrainThread()

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

A single thread invoked by the Train function.

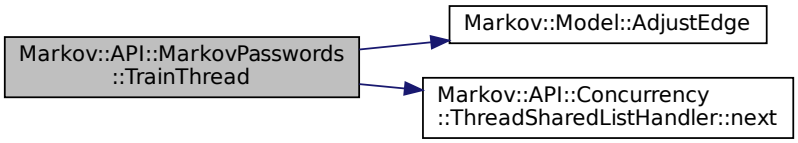
Parameters

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

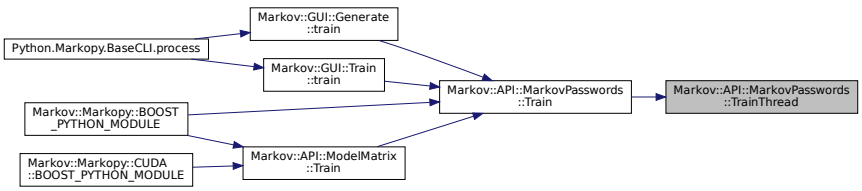
Definition at line 85 of file markovPasswords.cpp.

```
00085
                {
00086     char format_str[] ="%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
      "%ld,%s"
00097 #else
00098         sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100         this->AdjustEdge((const char*)linebuf, oc);
00101         delete linebuf;
00102     }
00103 }
```

References Markov::Model< NodeStorageType >::AdjustEdge(), keepRunning, and Markov::API::Concurrency::ThreadSharedListHa
Referenced by Markov::API::MarkovPasswords::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.10.4 Member Data Documentation

8.10.4.1 alternatingKernels

```
int Markov::API::CUDA::CUDAModelMatrix::alternatingKernels  [private], [inherited]
```
Definition at line 135 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk().

8.10.4.2 args

```
Python.CudaMarkopy.CudaMarkopyCLI.args
```
Definition at line 64 of file cudamarkopy.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.generate, Python.Markopy.MarkopyCLI.help(), Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.parse(), Python.CudaMarkopy.CudaMarkopyCLI.parse_fail(), Python.Markopy.BaseCLI.process(), and Python.Markopy.BaseCLI.train().

8.10.4.3 bInfinite

```
Python.CudaMarkopy.CudaModelMatrixCLI.bInfinite  [inherited]
```
Definition at line 73 of file cudammx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate().

8.10.4.4 cli

```
Python.CudaMarkopy.CudaMarkopyCLI.cli
```
Definition at line 101 of file cudamarkopy.py.
Referenced by Python.Markopy.MarkopyCLI.process().

8.10.4.5 cudaBlocks

```
int Markov::API::CUDA::CUDAModelMatrix::cudaBlocks  [private], [inherited]
```
Definition at line 125 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk().

8.10.4.6 cudaGridSize

```
int Markov::API::CUDA::CUDAModelMatrix::cudaGridSize  [private], [inherited]
```
Definition at line 131 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk(), and Markov::API::CUDA::CUDAModelMatrix::prepKerne

8.10.4.7 cudaMemPerGrid

```
int Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid  [private], [inherited]
```
Definition at line 132 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk(), and Markov::API::CUDA::CUDAModelMatrix::GatherAsy

8.10.4.8 cudaPerKernelAllocationSize

```
long int Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize  [private], [inherited]
```
Definition at line 133 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk(), Markov::API::CUDA::CUDAModelMatrix::GatherAsyncK
and Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel().

8.10.4.9 cudastreams

`cudaStream_t* Markov::API::CUDA::CUDAModelMatrix::cudastreams [private], [inherited]`
Definition at line 139 of file cudaModelMatrix.h.

8.10.4.10 cudaThreads

`int Markov::API::CUDA::CUDAModelMatrix::cudaThreads [private], [inherited]`
Definition at line 126 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk().

8.10.4.11 datasetFile

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

8.10.4.12 device_edgeMatrix

`char* Markov::API::CUDA::CUDAModelMatrix::device_edgeMatrix [private], [inherited]`
VRAM Address pointer of edge matrix (from modelMatrix.h)
Definition at line 88 of file cudaModelMatrix.h.

8.10.4.13 device_matrixIndex

`char* Markov::API::CUDA::CUDAModelMatrix::device_matrixIndex [private], [inherited]`
VRAM Address pointer of matrixIndex (from modelMatrix.h)
Definition at line 98 of file cudaModelMatrix.h.

8.10.4.14 device_outputBuffer

`char** Markov::API::CUDA::CUDAModelMatrix::device_outputBuffer [private], [inherited]`
RandomWalk results in device.
Definition at line 108 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel().

8.10.4.15 device_seeds

`unsigned long** Markov::API::CUDA::CUDAModelMatrix::device_seeds [private], [inherited]`
Definition at line 137 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel().

8.10.4.16 device_totalEdgeWeights

`long int* Markov::API::CUDA::CUDAModelMatrix::device_totalEdgeWeights [private], [inherited]`
VRAM Address pointer of total edge weights (from modelMatrix.h)
Definition at line 103 of file cudaModelMatrix.h.

8.10.4.17 device_valueMatrix

`long int* Markov::API::CUDA::CUDAModelMatrix::device_valueMatrix [private], [inherited]`
VRAM Address pointer of value matrix (from modelMatrix.h)
Definition at line 93 of file cudaModelMatrix.h.

8.10.4.18 edgeMatrix [1/3]

char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line 175 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.10.4.19 edgeMatrix [2/3]

char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line 175 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.10.4.20 edgeMatrix [3/3]

char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line 175 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.10.4.21 edges

std::vector<Edge<char >*> Markov::Model< char >::edges [private], [inherited]
A list of all edges in this model.
Definition at line 204 of file model.h.

8.10.4.22 fileIO [1/2]

Python.Markopy.ModelMatrixCLI.fileIO [inherited]
Definition at line 38 of file mmx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), and Python.Markopy.ModelMatrixCLI._generate().

8.10.4.23 fileIO [2/2]

Python.Markopy.ModelMatrixCLI.fileIO [inherited]
Definition at line 38 of file mmx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), and Python.Markopy.ModelMatrixCLI._generate().

8.10.4.24 flatEdgeMatrix

char* Markov::API::CUDA::CUDAModelMatrix::flatEdgeMatrix [private], [inherited]
Adding Edge matrix end-to-end and resize to 1-D array for better perfomance on traversing.
Definition at line 118 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix().

8.10.4.25 flatValueMatrix

long int* Markov::API::CUDA::CUDAModelMatrix::flatValueMatrix [private], [inherited]
Adding Value matrix end-to-end and resize to 1-D array for better perfomance on traversing.
Definition at line 123 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix().

8.10.4.26 iterationsPerKernelThread

int Markov::API::CUDA::CUDAModelMatrix::iterationsPerKernelThread [private], [inherited]
Definition at line 127 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk().

8.10.4.27 matrixIndex [1/3]

char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]
to hold the Matrix index (To hold the orders of 2-D arrays')
Definition at line 190 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.10.4.28 matrixIndex [2/3]

char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]
to hold the Matrix index (To hold the orders of 2-D arrays')
Definition at line 190 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.10.4.29 matrixIndex [3/3]

char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]
to hold the Matrix index (To hold the orders of 2-D arrays')
Definition at line 190 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.10.4.30 matrixSize [1/3]

int Markov::API::ModelMatrix::matrixSize [protected], [inherited]
to hold Matrix size
Definition at line 185 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), Markov::API::ModelMatrix::FastRandomWalkThread(), and Markov::API::CUDA::CUDAMod

8.10.4.31 matrixSize [2/3]

int Markov::API::ModelMatrix::matrixSize [protected], [inherited]
to hold Matrix size
Definition at line 185 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), Markov::API::ModelMatrix::FastRandomWalkThread(), and Markov::API::CUDA::CUDAMod

8.10.4.32 matrixSize [3/3]

`int Markov::API::ModelMatrix::matrixSize [protected], [inherited]`
to hold Matrix size
Definition at line 185 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), Markov::API::ModelMatrix::FastRandomWalkThread(), and Markov::API::CUDA::CUDAMod

8.10.4.33 model [1/4]

`Python.CudaMarkopy.CudaModelMatrixCLI.model [inherited]`
Definition at line 65 of file cudammx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.10.4.34 model [2/4]

`Python.Markopy.BaseCLI.model [inherited]`
Definition at line 40 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.10.4.35 model [3/4]

`Python.Markopy.ModelMatrixCLI.model [inherited]`
Definition at line 30 of file mmx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.10.4.36 model [4/4]

`Python.Markopy.MarkovPasswordsCLI.model [inherited]`
Definition at line 29 of file mp.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.10.4.37 modelSavefile

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

Definition at line 124 of file markovPasswords.h.

8.10.4.38 nodes

`std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]`
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.

8.10.4.39 numberOfPartitions

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

8.10.4.40 outputBuffer

`char** Markov::API::CUDA::CUDAModelMatrix::outputBuffer [private], [inherited]`
RandomWalk results in host.
Definition at line 113 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput(), and Markov::API::CUDA::CUDAModelMatrix::pr

8.10.4.41 outputFile

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

Definition at line 125 of file markovPasswords.h.

8.10.4.42 parser [1/6]

`Python.Markopy.BaseCLI.parser [inherited]`
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.10.4.43 parser [2/6]

`Python.Markopy.BaseCLI.parser [inherited]`
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.10.4.44 parser [3/6]

`Python.Markopy.BaseCLI.parser [inherited]`
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.10.4.45 parser [4/6]

`Python.Markopy.BaseCLI.parser [inherited]`
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(),

Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.10.4.46 parser [5/6]

Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.10.4.47 parser [6/6]

Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.10.4.48 print_help [1/6]

Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.10.4.49 print_help [2/6]

Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.10.4.50 print_help [3/6]

Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.10.4.51 print_help [4/6]

Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.10.4.52 print_help [5/6]

Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.10.4.53 print_help [6/6]

Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.10.4.54 ready [1/3]

bool Markov::API::ModelMatrix::ready [protected], [inherited]
True when matrix is constructed. False if not.
Definition at line 200 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::ModelMatrix().

8.10.4.55 ready [2/3]

bool Markov::API::ModelMatrix::ready [protected], [inherited]
True when matrix is constructed. False if not.
Definition at line 200 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::ModelMatrix().

8.10.4.56 ready [3/3]

bool Markov::API::ModelMatrix::ready [protected], [inherited]
True when matrix is constructed. False if not.
Definition at line 200 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::ModelMatrix().

8.10.4.57 starterNode

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

8.10.4.58 totalEdgeWeights [1/3]

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]
Array of the Total Edge Weights.
Definition at line 195 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.10.4.59 totalEdgeWeights [2/3]

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]
Array of the Total Edge Weights.
Definition at line 195 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.10.4.60 totalEdgeWeights [3/3]

`long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]`
Array of the Total Edge Weights.
Definition at line 195 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.10.4.61 totalOutputPerKernel

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

8.10.4.62 totalOutputPerSync

`long int Markov::API::CUDA::CUDAModelMatrix::totalOutputPerSync [private], [inherited]`
Definition at line 128 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk().

8.10.4.63 valueMatrix [1/3]

`long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]`
2-d Integer array for the value Matrix (For the weights of Edges)
Definition at line 180 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.10.4.64 valueMatrix [2/3]

`long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]`
2-d Integer array for the value Matrix (For the weights of Edges)
Definition at line 180 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.10.4.65 valueMatrix [3/3]

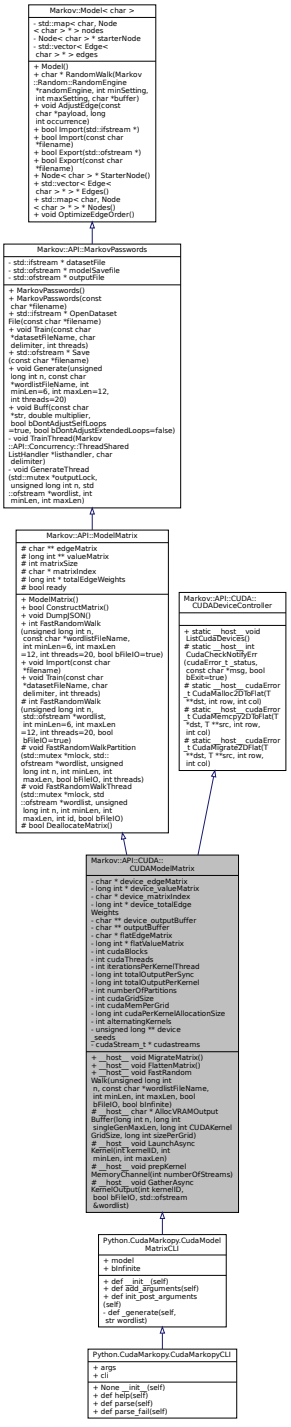
`long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]`
2-d Integer array for the value Matrix (For the weights of Edges)
Definition at line 180 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().
The documentation for this class was generated from the following file:

- Markopy/CudaMarkopy/src/CLI/cudamarkopy.py

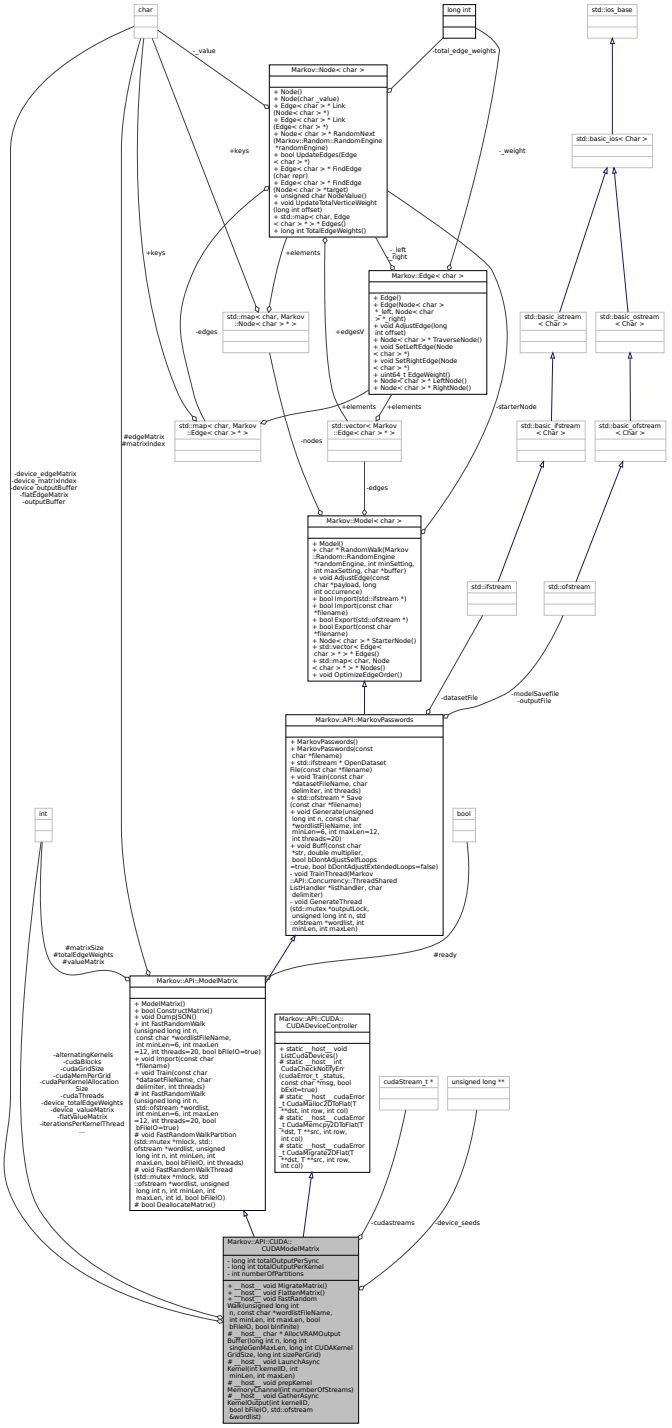
8.11 Markov::API::CUDA::CUDAModelMatrix Class Reference

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

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



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



Public Member Functions

- __host__ void MigrateMatrix ()

  Migrate the class members to the VRAM.

- __host__ void FlattenMatrix ()

  Flatten migrated matrix from 2d to 1d.

- __host__ void FastRandomWalk (unsigned long int n, const char ∗wordlistFileName, int minLen, int maxLen, bool bFileIO, bool bInfinite)

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

- bool ConstructMatrix ()

  Construct the related Matrix data for the model.

- void DumpJSON ()

  Debug function to dump the model to a JSON file.

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

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

- void Import (const char *filename)

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

- bool Import (std::ifstream *)

  Import a file to construct the model.

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

  Train the model with the dataset file.

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

  Open dataset file and return the ifstream pointer.

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

  Export model to file.

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

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

- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.

- char * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

  Do a random walk on this model.

- void AdjustEdge (const char *payload, long int occurrence)

  Adjust the model with a single string.

- bool Export (std::ofstream *)

  Export a file of the model.

- bool Export (const char *filename)

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

- Node< char > * StarterNode ()

  Return starter Node.

- std::vector< Edge< char > * > * Edges ()

  Return a vector of all the edges in the model.

- std::map< char, Node< char > * > * Nodes ()

  Return starter Node.

- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.

Static Public Member Functions

- static __host__ void ListCudaDevices ()

  List CUDA devices in the system.

Protected Member Functions

- __host__ char * AllocVRAMOutputBuffer (long int n, long int singleGenMaxLen, long int CUDAKernelGrid↩ Size, long int sizePerGrid)

  Allocate the output buffer for kernel operation.
- __host__ void LaunchAsyncKernel (int kernelID, int minLen, int maxLen)
- __host__ void prepKernelMemoryChannel (int numberOfStreams)
- __host__ void GatherAsyncKernelOutput (int kernelID, bool bFileIO, std::ofstream &wordlist)
- int FastRandomWalk (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- void FastRandomWalkPartition (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

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

  A single thread of a single partition of FastRandomWalk.
- bool DeallocateMatrix ()

  Deallocate matrix and make it ready for re-construction.

Static Protected Member Functions

- static __host__ int CudaCheckNotifyErr (cudaError_t _status, const char *msg, bool bExit=true)

  Check results of the last operation on GPU.
- template<typename T >
  static __host__ cudaError_t CudaMalloc2DToFlat (T **dst, int row, int col)

  Malloc a 2D array in device space.
- template<typename T >
  static __host__ cudaError_t CudaMemcpy2DToFlat (T *dst, T **src, int row, int col)

  Memcpy a 2D array in device space after flattening.
- template<typename T >
  static __host__ cudaError_t CudaMigrate2DFlat (T **dst, T **src, int row, int col)

  Both malloc and memcpy a 2D array into device VRAM.

Protected Attributes

- char ** edgeMatrix

  2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** valueMatrix

  2-d Integer array for the value Matrix (For the weights of Edges)
- int matrixSize

  to hold Matrix size
- char * matrixIndex

  to hold the Matrix index (To hold the orders of 2-D arrays')
- long int * totalEdgeWeights

  Array of the Total Edge Weights.
- bool ready

  True when matrix is constructed. False if not.

Private Member Functions

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

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

  A single thread invoked by the Generate function.

Private Attributes

- char * device_edgeMatrix

  VRAM Address pointer of edge matrix (from modelMatrix.h)
- long int * device_valueMatrix

  VRAM Address pointer of value matrix (from modelMatrix.h)
- char * device_matrixIndex

  VRAM Address pointer of matrixIndex (from modelMatrix.h)
- long int * device_totalEdgeWeights

  VRAM Address pointer of total edge weights (from modelMatrix.h)
- char ** device_outputBuffer

  RandomWalk results in device.
- char ** outputBuffer

  RandomWalk results in host.
- char * flatEdgeMatrix

  Adding Edge matrix end-to-end and resize to 1-D array for better perfomance on traversing.
- long int * flatValueMatrix

  Adding Value matrix end-to-end and resize to 1-D array for better perfomance on traversing.
- int cudaBlocks
- int cudaThreads
- int iterationsPerKernelThread
- long int totalOutputPerSync
- long int totalOutputPerKernel
- int numberOfPartitions
- int cudaGridSize
- int cudaMemPerGrid
- long int cudaPerKernelAllocationSize
- int alternatingKernels
- unsigned long ** device_seeds
- cudaStream_t * cudastreams
- std::ifstream * datasetFile
- std::ofstream * modelSavefile

  Dataset file input of our system
- std::ofstream * outputFile

  File to save model of our system
- std::map< char, Node< char > * > nodes

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

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

  A list of all edges in this model.

8.11.1 Detailed Description

Extension of Markov::API::ModelMatrix which is modified to run on GPU devices.
This implementation only supports Nvidia devices.
Class to flatten and reduce Markov::Model to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)
To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are sychronized and files are flushed every 50M operations.
Definition at line 19 of file cudaModelMatrix.h.

8.11.2 Member Function Documentation

8.11.2.1 AdjustEdge()

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

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

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

Parameters

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

Definition at line 109 of file model.h.

```
00337                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.11.2.2 AllocVRAMOutputBuffer()

```
__host__ char* Markov::API::CUDA::CUDAModelMatrix::AllocVRAMOutputBuffer (
            long int n,
            long int singleGenMaxLen,
            long int CUDAKernelGridSize,
            long int sizePerGrid )  [protected]
```

Allocate the output buffer for kernel operation.
TODO

Parameters

n	- Number of passwords to generate.
singleGenMaxLen	- maximum string length for a single generation
CUDAKernelGridSize	- Total number of grid members in CUDA kernel
sizePerGrid	- Size to allocate per grid member

Returns

pointer to the allocation on VRAM

8.11.2.3 Buff()

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

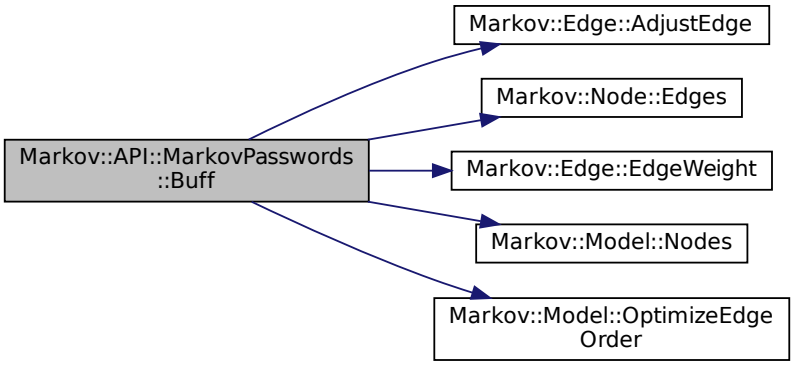
Definition at line 153 of file markovPasswords.cpp.

```
00153                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173
00174         }
00175         i++;
00176     }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

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

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.11.2.4 ConstructMatrix()

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

Construct the related Matrix data for the model.
This operation can be used after importing/training to allocate and populate the matrix content.
this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

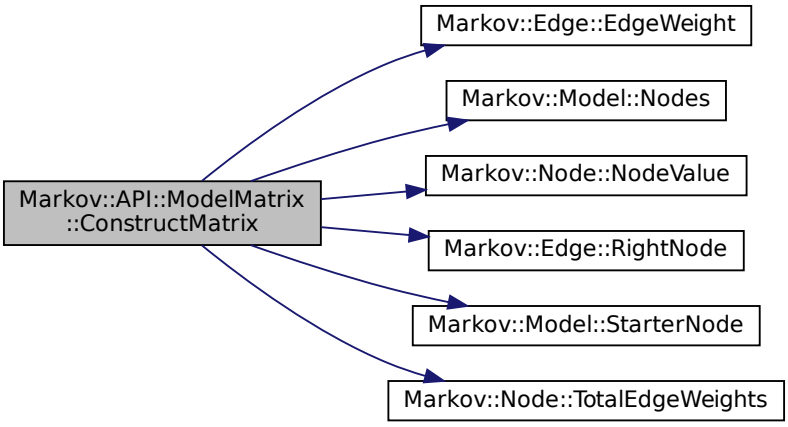
True if constructed. False if already construced.

Definition at line 31 of file modelMatrix.cpp.

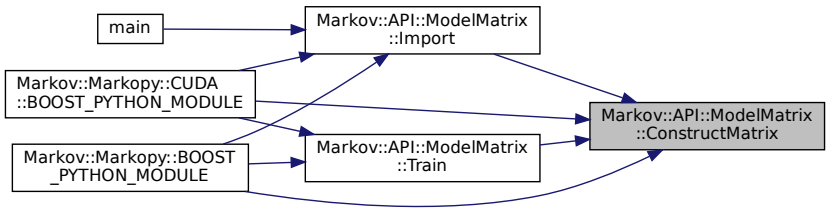
```
00031                                       {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034 
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037 
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
```

```
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072
00073         }
00074         i++;
00075     }
00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::API::ModelMatrix::r
Markov::API::ModelMatrix::matrixSize, Markov::Model< NodeStorageType >::Nodes(), Markov::Node< storageType >::NodeValue()
Markov::API::ModelMatrix::ready, Markov::Edge< NodeStorageType >::RightNode(), Markov::Model< NodeStorageType >::StarterN
Markov::API::ModelMatrix::totalEdgeWeights, Markov::Node< storageType >::TotalEdgeWeights(), and Markov::API::ModelMatrix::va
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(),
Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.11.2.5 CudaCheckNotifyErr()

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

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

Parameters

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

Returns

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

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

Definition at line 32 of file cudaDeviceController.cu.

```
00032
                    {
00033         if (_status != cudaSuccess) {
00034             std::cerr « "\033[1;31m" « msg  « " -> " « cudaGetErrorString(_status)  « " ("« _status «
      ")" « "\033[0m" « "\n";
00035
00036             if(bExit) {
00037                 cudaDeviceReset();
00038                 exit(1);
00039             }
00040         }
00041         return 0;
00042     }
```

8.11.2.6 CudaMalloc2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat (
            T ** dst,
            int row,
            int col )  [inline], [static], [protected], [inherited]
```

Malloc a 2D array in device space.
This function will allocate enough space on VRAM for flattened 2D array.

Parameters

dst	destination pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

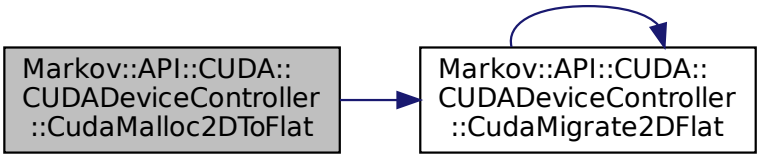
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess){
    CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
}
```

Definition at line 75 of file cudaDeviceController.h.

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

References Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().

Here is the call graph for this function:



8.11.2.7 CudaMemcpy2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat (
            T * dst,
            T ** src,
            int row,
            int col )  [inline], [static], [protected], [inherited]
```

Memcpy a 2D array in device space after flattening.

Resulting buffer will not be true 2D array.

Parameters

dst	destination pointer
rc	source pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

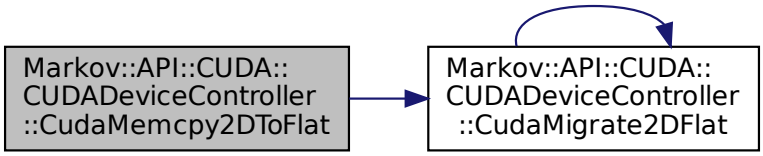
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
CudaCheckNotifyErr(cudastatus, "  CudaMemcpy2DToFlat Failed.", false);
```

Definition at line 103 of file cudaDeviceController.h.

```
00103                                                                                {
00104             T* tempbuf = new T[row*col];
00105             for(int i=0;i<row;i++){
00106                 memcpy(&(tempbuf[row*i]), src[i], col);
00107             }
00108             return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00109
00110         }
```

References Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().

Here is the call graph for this function:



8.11.2.8 CudaMigrate2DFlat()

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

Both malloc and memcpy a 2D array into device VRAM.

Resulting buffer will not be true 2D array.

Parameters

dst	destination pointer
rc	source pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMigrate2DFlat<long int>(
   &dst, this->valueMatrix, this->matrixSize, this->matrixSize);
CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
```

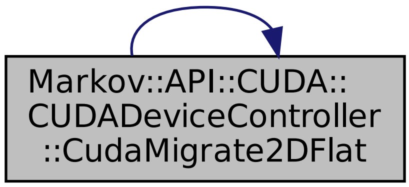
Definition at line 132 of file cudaDeviceController.h.

```
00132                                                                                 {
00133             cudaError_t cudastatus;
00134             cudastatus = CudaMalloc2DToFlat<T>(dst, row, col);
00135             if(cudastatus!=cudaSuccess){
00136                 CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
00137                 return cudastatus;
00138             }
00139             cudastatus = CudaMemcpy2DToFlat<T>(*dst,src,row,col);
00140             CudaCheckNotifyErr(cudastatus, "  CudaMemcpy2DToFlat Failed.", false);
00141             return cudastatus;
00142         }
```

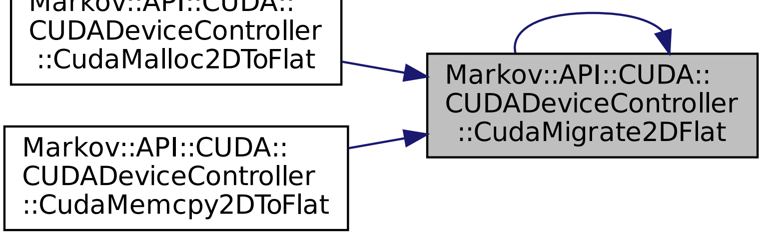
References Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().

Referenced by Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat(), Markov::API::CUDA::CUDADeviceController::Cu and Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().

Here is the call graph for this function:



Here is the caller graph for this function:



8.11.2.9 DeallocateMatrix()

`bool Markov::API::ModelMatrix::DeallocateMatrix ( )  [protected], [inherited]`

Deallocate matrix and make it ready for re-construction.

Returns

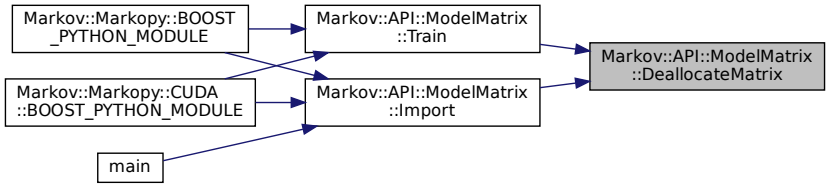
True if deallocated. False if matrix was not initialized

Definition at line 81 of file modelMatrix.cpp.

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

```
00085
00086     for(int i=0;i<this->matrixSize;i++){
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090
00091     for(int i=0;i<this->matrixSize;i++){
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
00099 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, Markov::API::ModelMatrix::ready, Markov::API::ModelMatrix::totalEdgeWeights, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().
Here is the caller graph for this function:



8.11.2.10 DumpJSON()

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

Debug function to dump the model to a JSON file.
Might not work 100%. Not meant for production use.
Definition at line 101 of file modelMatrix.cpp.

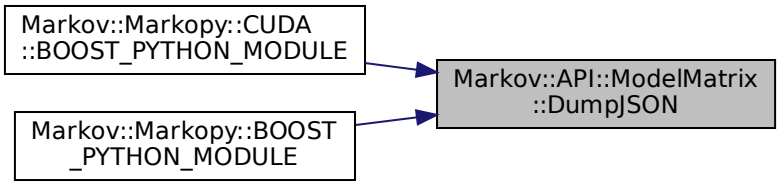
```
00101                                   {
00102
00103     std::cout « "{\n   \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout « "\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout « "\\\\";
00107         else if(this->matrixIndex[i]==0) std::cout « "\\\\x00";
00108         else if(i==0) std::cout « "\\\\xff";
00109         else if(this->matrixIndex[i]=='\n') std::cout « "\\n";
00110         else std::cout « this->matrixIndex[i];
00111     }
00112     std::cout «
00113     "\",\n"
00114     "   \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++){
00117         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00120         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00121         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00122         for(int j=0;j<this->matrixSize;j++){
00123             if(this->edgeMatrix[i][j]=='"') std::cout « "\"\\\"\"";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout « "\"\\\\\"";
00125             else if(this->edgeMatrix[i][j]==0) std::cout « "\"\\\\x00\"";
00126             else if(this->edgeMatrix[i][j]<0) std::cout « "\"\\\\xff\"";
00127             else if(this->matrixIndex[i]=='\n') std::cout « "\"\\n\"";
00128             else std::cout « "\"" « this->edgeMatrix[i][j] « "\"";
00129             if(j!=this->matrixSize-1) std::cout « ", ";
00130         }
00131         std::cout « "],\n";
00132     }
00133     std::cout « "},\n";
00134
00135     std::cout « "\"   weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++){
```

```
00137         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00140         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00141         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++){
00144             std::cout « this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout « ", ";
00146         }
00147         std::cout « "],\n";
00148     }
00149     std::cout « "  }\n}\n";
00150 }
```

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

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

Here is the caller graph for this function:



8.11.2.11 Edges()

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

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.11.2.12 Export() [1/2]

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

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

Returns

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

Example Use: Export file to filename

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

Definition at line 166 of file model.h.

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

8.11.2.13 Export() [2/2]

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

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

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

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

Definition at line 155 of file model.h.

```
00288                                                    {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
      e->RightNode()->NodeValue() << "\n";
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.11.2.14 FastRandomWalk() [1/3]

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

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

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

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

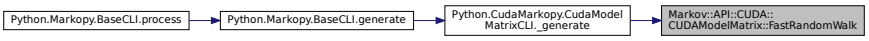
Definition at line 58 of file cudaModelMatrix.cu.

```
00058
                                                                 {
00059         cudaDeviceProp prop;
00060         int device=0;
00061         cudaGetDeviceProperties(&prop, device);
00062         cudaChooseDevice(&device, &prop);
00063         //std::cout << "Flattening matrix." << std::endl;
00064         this->FlattenMatrix();
00065         //std::cout << "Migrating matrix." << std::endl;
00066         this->MigrateMatrix();
00067         //std::cout << "Migrated matrix." << std::endl;
00068         std::ofstream wordlist;
```

```
00069         if(bFileIO)
00070             wordlist.open(wordlistFileName);
00071 
00072 
00073         cudaBlocks = 1024;
00074         cudaThreads = 256;
00075         iterationsPerKernelThread = 100;
00076         alternatingKernels = 2;
00077         totalOutputPerKernel = (long int)cudaBlocks*(long int)cudaThreads*iterationsPerKernelThread;
00078         totalOutputPerSync= totalOutputPerKernel*alternatingKernels;
00079         numberOfPartitions = n/totalOutputPerSync;
00080         cudaGridSize = cudaBlocks*cudaThreads;
00081         cudaMemPerGrid = (maxLen+2)*iterationsPerKernelThread;
00082         cudaPerKernelAllocationSize = cudaGridSize*cudaMemPerGrid;
00083         this->prepKernelMemoryChannel(alternatingKernels);
00084 
00085         unsigned long int leftover = n - (totalOutputPerSync*numberOfPartitions);
00086 
00087         if(bInfinite && !numberOfPartitions) numberOfPartitions=5;
00088         std::cerr << cudaPerKernelAllocationSize << "\n";
00089 
00090         if(n%totalOutputPerSync) std::cerr << "For optimization, request outputs muliples of "<<
      totalOutputPerSync << ".\n";
00091 
00092         //start kernelID 1
00093         this->LaunchAsyncKernel(1, minLen, maxLen);
00094 
00095         for(int i=1;i<numberOfPartitions;i++){
00096             if(bInfinite) i=0;
00097 
00098             //wait kernelID1 to finish, and start kernelID 0
00099             cudaStreamSynchronize(this->cudastreams[1]);
00100             this->LaunchAsyncKernel(0, minLen, maxLen);
00101 
00102             //start memcpy from kernel 1 (block until done)
00103             this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00104 
00105             //wait kernelID 0 to finish, then start kernelID1
00106             cudaStreamSynchronize(this->cudastreams[0]);
00107             this->LaunchAsyncKernel(1, minLen, maxLen);
00108 
00109             //start memcpy from kernel 0 (block until done)
00110             this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00111 
00112         }
00113 
00114         //wait kernelID1 to finish, and start kernelID 0
00115         cudaStreamSynchronize(this->cudastreams[1]);
00116         this->LaunchAsyncKernel(0, minLen, maxLen);
00117         this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00118         cudaStreamSynchronize(this->cudastreams[0]);
00119         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00120 
00121 
00122         if(!leftover) return;
00123         alternatingKernels=1;
00124         std::cerr << "Remaining line count (" << leftover << ") is lower than partition. Adjusting CUDA
      workload..\n";
00125         this->iterationsPerKernelThread = leftover/cudaGridSize;
00126         this->LaunchAsyncKernel(0, minLen, maxLen);
00127         cudaStreamSynchronize(this->cudastreams[0]);
00128         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00129 
00130         leftover -= this->iterationsPerKernelThread*cudaGridSize;
00131         if(!leftover) return;
00132 
00133         std::cerr << "Remaining line count (" << leftover << ") is lower than minimum possible. Handing
      over to CPU generation.\n";
00134         this->iterationsPerKernelThread = leftover/cudaGridSize;
00135 
00136         leftover -= this->iterationsPerKernelThread;
00137 
00138         if(!leftover) return;
00139         std::cerr << "Remaining " << leftover << " lines are absolutely not worth printing.\n";
00140         Markov::API::ModelMatrix::ConstructMatrix();
00141         Markov::API::ModelMatrix::FastRandomWalk(leftover, &wordlist, minLen, maxLen, 1, bFileIO);
00142 
00143     }
```

References alternatingKernels, cudaBlocks, cudaGridSize, cudaMemPerGrid, cudaPerKernelAllocationSize, cudaThreads, iterationsPerKernelThread, numberOfPartitions, totalOutputPerKernel, and totalOutputPerSync.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI::_generate().

Here is the caller graph for this function:



8.11.2.15 FastRandomWalk() [2/3]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [inherited]
```

Random walk on the Matrix-reduced Markov::Model.
This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.
While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

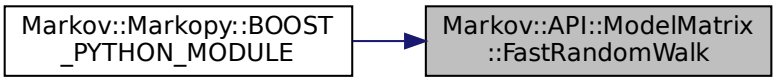
```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 217 of file modelMatrix.cpp.

```
00217
                                          {
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

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

Here is the caller graph for this function:



8.11.2.16 FastRandomWalk() [3/3]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [protected], [inherited]
```

Random walk on the Matrix-reduced Markov::Model.
This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.
While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

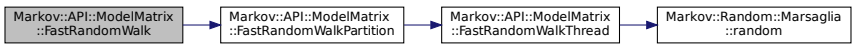
```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 204 of file modelMatrix.cpp.

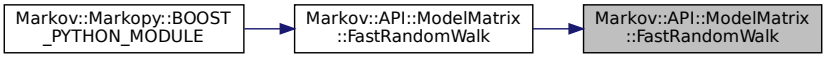
```
00204                                   {
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
      threads);
00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
      threads);
00213     }
00214     return 0;
00215 }
```

References Markov::API::ModelMatrix::FastRandomWalkPartition().
Referenced by Markov::API::ModelMatrix::FastRandomWalk().

Here is the call graph for this function:



Here is the caller graph for this function:



8.11.2.17 FastRandomWalkPartition()

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

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

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

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

Parameters

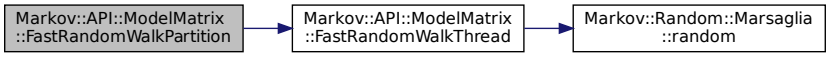
mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads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

Definition at line 225 of file modelMatrix.cpp.

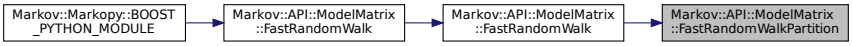
```
00225
                                                                {
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
```

```
00232     int id = 0;
00233     for(int i=0;i<threads;i++){
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
      mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
      wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00239
00240     for(int i=0;i<threads;i++){
00241         threadsV[i]->join();
00242     }
00243 }
```

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



Here is the caller graph for this function:



8.11.2.18 FastRandomWalkThread()

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

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

Parameters

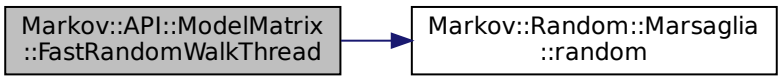
mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
id	- **DEPRECATED** Thread id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

Definition at line 153 of file modelMatrix.cpp.

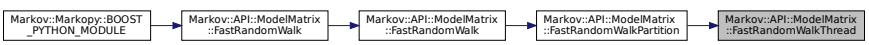
```
00153
```

```
                                                    {
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179
00180             if (len >= maxLen)  break;
00181             else if ((next < 0) && (len < minLen)) continue;
00182             else if (next < 0) break;
00183             cur = next;
00184             res[bufferctr + len++] = cur;
00185         }
00186         res[bufferctr + len++] = '\n';
00187         bufferctr+=len;
00188
00189     }
00190     if(bFileIO){
00191         mlock->lock();
00192         *wordlist << res;
00193         mlock->unlock();
00194     }else{
00195         mlock->lock();
00196         std::cout << res;
00197         mlock->unlock();
00198     }
00199     delete res;
00200
00201 }
```

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



Here is the caller graph for this function:



8.11.2.19 FlattenMatrix()

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

Flatten migrated matrix from 2d to 1d.
Definition at line 261 of file cudaModelMatrix.cu.

```
00261                                                   {
00262         this->flatEdgeMatrix = new char[this->matrixSize*this->matrixSize];
00263
00264         this->flatValueMatrix = new long int[this->matrixSize*this->matrixSize];
00265         for(int i=0;i<this->matrixSize;i++){
00266             memcpy(&this->flatEdgeMatrix[i*this->matrixSize], this->edgeMatrix[i], this->matrixSize );
00267             memcpy(&this->flatValueMatrix[i*this->matrixSize], this->valueMatrix[i],
       this->matrixSize*sizeof(long int) );
00268         }
00269     }
```

References flatEdgeMatrix, flatValueMatrix, and Markov::API::ModelMatrix::matrixSize.

8.11.2.20 GatherAsyncKernelOutput()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput (
             int kernelID,
             bool bFileIO,
             std::ofstream & wordlist )  [protected]
```

Definition at line 180 of file cudaModelMatrix.cu.

```
00180                                                   {
00181
       cudaMemcpy(this->outputBuffer[kernelID],this->device_outputBuffer[kernelID],cudaPerKernelAllocationSize,
       cudaMemcpyDeviceToHost);
00182         //std::cerr << "Kernel" << kernelID << " output copied\n";
00183         if(bFileIO){
00184             for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00185                 wordlist << &this->outputBuffer[kernelID][j];
00186             }
00187         }else{
00188             for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00189                 std::cout << &this->outputBuffer[kernelID][j];
00190             }
00191         }
00192     }
```

References cudaMemPerGrid, cudaPerKernelAllocationSize, and outputBuffer.

8.11.2.21 Generate()

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

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

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

Parameters

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

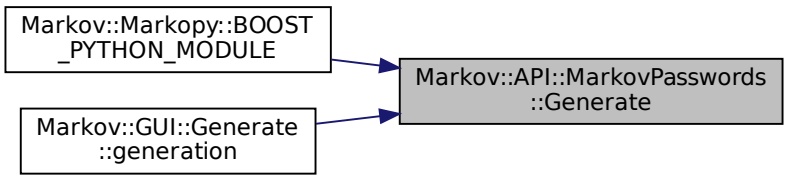
Definition at line 118 of file markovPasswords.cpp.

```
00118                                   {
00119     char* res;
```

```
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
      &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

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

Here is the caller graph for this function:



8.11.2.22 GenerateThread()

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

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

Parameters

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

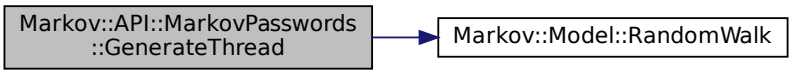
Parameters

minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

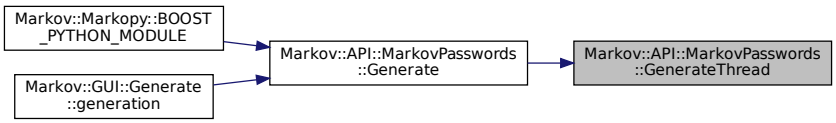
Definition at line 140 of file markovPasswords.cpp.

```
00140                                       {
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist « res « "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

References Markov::Model< NodeStorageType >::RandomWalk().
Referenced by Markov::API::MarkovPasswords::Generate().
Here is the call graph for this function:



Here is the caller graph for this function:



8.11.2.23 Import() [1/2]

```
void Markov::API::ModelMatrix::Import (
            const char * filename )  [inherited]
```

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

Returns

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

Example Use: Import a file with filename

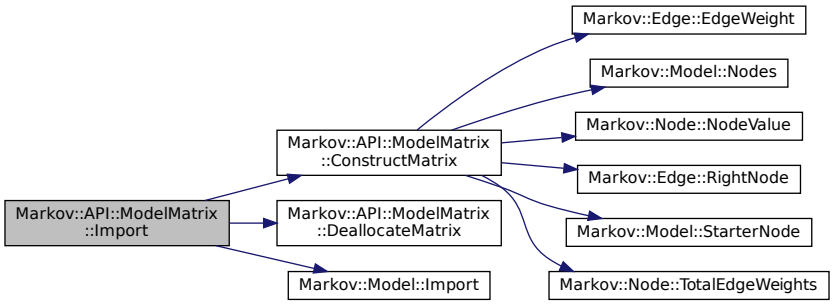
```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.
Definition at line 19 of file modelMatrix.cpp.

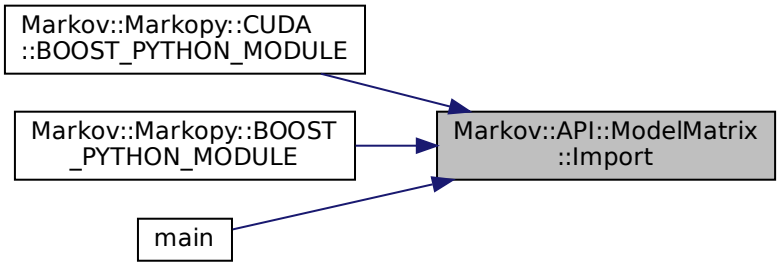
```
00019                                                   {
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

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

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



Here is the caller graph for this function:



8.11.2.24 Import() [2/2]

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

Import a file to construct the model.
File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr
Iterate over this list, and construct nodes and edges accordingly.

Returns

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

Example Use: Import a file from ifstream

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

Definition at line 126 of file model.h.

```
00216                                                   {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
```

```
00224         //std::cout « "cell: " « cell « std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout « oc « "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout « "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout « "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

8.11.2.25 LaunchAsyncKernel()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::LaunchAsyncKernel (
              int kernelID,
              int minLen,
              int maxLen )  [protected]
```

Definition at line 171 of file cudaModelMatrix.cu.

```
00171                                                                                                 {
00172
00173         //if(kernelID == 0);// cudaStreamSynchronize(this->cudastreams[2]);
00174         //else cudaStreamSynchronize(this->cudastreams[kernelID-1]);
00175         FastRandomWalkCUDAKernel«<cudaBlocks,cudaThreads,0,
      this->cudastreams[kernelID]»>(iterationsPerKernelThread, minLen, maxLen,
      this->device_outputBuffer[kernelID], this->device_matrixIndex,
00176             this->device_totalEdgeWeights, this->device_valueMatrix, this->device_edgeMatrix,
      this->matrixSize, cudaMemPerGrid, this->device_seeds[kernelID]);
00177         //std::cerr « "Started kernel" « kernelID « "\n";
00178     }
```

8.11.2.26 ListCudaDevices()

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

List CUDA devices in the system.

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

Example output:

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

Definition at line 16 of file cudaDeviceController.cu.

```
00016                                                                 { //list cuda Capable devices on
      host.
00017         int nDevices;
00018         cudaGetDeviceCount(&nDevices);
```

```
00019         for (int i = 0; i < nDevices; i++) {
00020             cudaDeviceProp prop;
00021             cudaGetDeviceProperties(&prop, i);
00022             std::cerr << "Device Number: " <<  i  << "\n";
00023             std::cerr << "Device name: " << prop.name << "\n";
00024             std::cerr << "Memory Clock Rate (KHz): " << prop.memoryClockRate << "\n";
00025             std::cerr << "Memory Bus Width (bits): " << prop.memoryBusWidth << "\n";
00026             std::cerr << "Peak Memory Bandwidth (GB/s): " << 2.0 * prop.memoryClockRate *
      (prop.memoryBusWidth / 8) / 1.0e6 << "\n";
00027             std::cerr << "Max Linear Threads: " << prop.maxThreadsDim[0] << "\n";
00028
00029         }
00030     }
```

8.11.2.27 MigrateMatrix()

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

Migrate the class members to the VRAM.

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

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

Definition at line 20 of file cudaModelMatrix.cu.

```
00020                                                   {
00021         cudaError_t cudastatus;
00022
00023         cudastatus = cudaMalloc((char**)&(this->device_matrixIndex),
00024             this->matrixSize*sizeof(char));
00025         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_matrixIndex.");
00026
00027         cudastatus = cudaMalloc((long int **)&(this->device_totalEdgeWeights),
      this->matrixSize*sizeof(long int));
00028         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_totalEdgeWeights.");
00029
00030         cudastatus = cudaMemcpy(this->device_matrixIndex, this->matrixIndex,
00031             this->matrixSize*sizeof(char), cudaMemcpyHostToDevice);
00032         CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Index)");
00033
00034         cudastatus = cudaMemcpy(this->device_totalEdgeWeights, this->totalEdgeWeights,
00035             this->matrixSize*sizeof(long int), cudaMemcpyHostToDevice);
00036         CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Total Edge Values)");
00037
00038         cudastatus = CudaMigrate2DFlat<char>(
00039             &(this->device_edgeMatrix), this->edgeMatrix, this->matrixSize, this->matrixSize);
00040         CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize edge matrix.");
00041
00042         cudastatus = CudaMigrate2DFlat<long int>(
00043             &(this->device_valueMatrix), this->valueMatrix, this->matrixSize, this->matrixSize);
00044         CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
00045
00046     }
```

8.11.2.28 Nodes()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.11.2.29 OpenDatasetFile()

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

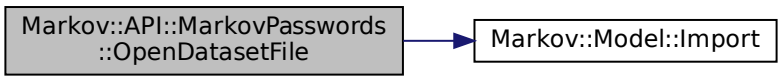
ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

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

Here is the call graph for this function:



8.11.2.30 OptimizeEdgeOrder()

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

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.11.2.31 prepKernelMemoryChannel()

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

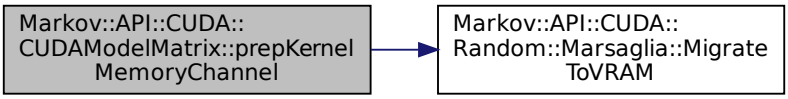
Definition at line 145 of file cudaModelMatrix.cu.

```
00145                                                                                {
00146
00147         this->cudastreams = new cudaStream_t[numberOfStreams];
00148         for(int i=0;i<numberOfStreams;i++)
00149             cudaStreamCreate(&this->cudastreams[i]);
00150
00151         this-> outputBuffer = new char*[numberOfStreams];
00152         for(int i=0;i<numberOfStreams;i++)
00153             this->outputBuffer[i]= new char[cudaPerKernelAllocationSize];
00154
```

```
00155         cudaError_t cudastatus;
00156         this-> device_outputBuffer = new char*[numberOfStreams];
00157             for(int i=0;i<numberOfStreams;i++){
00158                 cudastatus = cudaMalloc((char**)&(device_outputBuffer[i]),
      cudaPerKernelAllocationSize);
00159                 CudaCheckNotifyErr(cudastatus, "Failed to establish memory channel. Possibly out of
      VRAM?");
00160             }
00161
00162         this-> device_seeds = new unsigned long*[numberOfStreams];
00163         for(int i=0;i<numberOfStreams;i++){
00164             Markov::API::CUDA::Random::Marsaglia *MEarr = new
      Markov::API::CUDA::Random::Marsaglia[cudaGridSize];
00165             this->device_seeds[i] = Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(MEarr,
      cudaGridSize);
00166             delete[] MEarr;
00167         }
00168
00169     }
```

References cudaGridSize, cudaPerKernelAllocationSize, device_outputBuffer, device_seeds, Markov::API::CUDA::Random::Marsagli and outputBuffer.

Here is the call graph for this function:



8.11.2.32 RandomWalk()

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307
                                                        {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue();
00327     }
00328
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.11.2.33 Save()

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

Export model to file.

Parameters

filename	- Export filename.

Returns

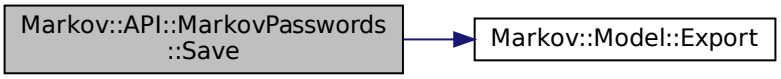
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                    {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

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

Here is the call graph for this function:



8.11.2.34 StarterNode()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.11.2.35 Train()

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

Train the model with the dataset file.

Parameters

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

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

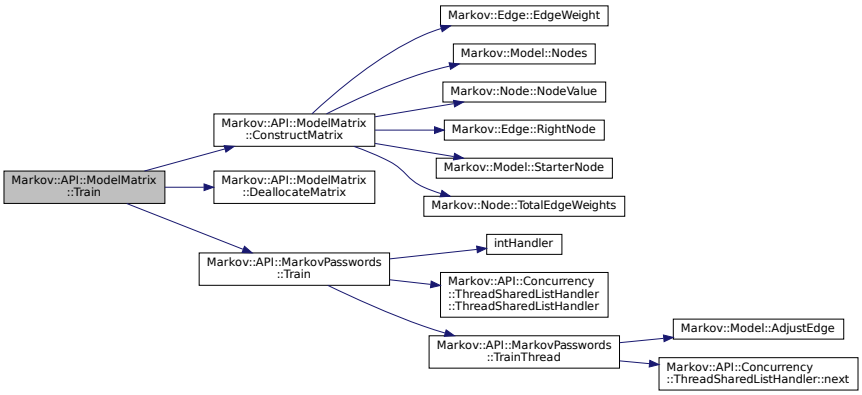
Construct the matrix when done.

Definition at line 25 of file modelMatrix.cpp.

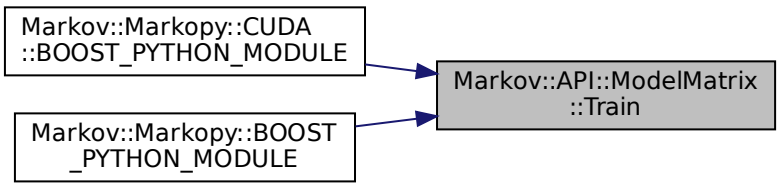
```
00025                                                                    {
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and

Markov::API::MarkovPasswords::Train().
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), and Markov::Markopy::BOOST_PYTHON_MODULE().
Here is the call graph for this function:



Here is the caller graph for this function:



8.11.2.36 TrainThread()

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

A single thread invoked by the Train function.

Parameters

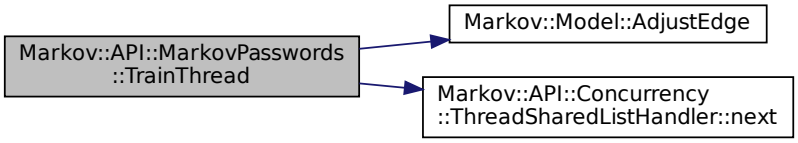
listhandler	- Listhandler class to read corpus from
delimiter	- a character, same as the delimiter in dataset content

Definition at line 85 of file markovPasswords.cpp.

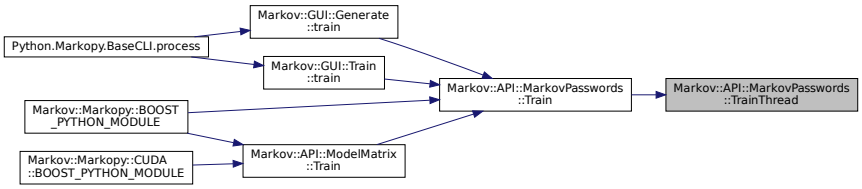
```
00085
                {
00086     char format_str[] ="%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
```

```
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
      "%ld,%s"
00097 #else
00098         sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100         this->AdjustEdge((const char*)linebuf, oc);
00101         delete linebuf;
00102     }
00103 }
```

References Markov::Model< NodeStorageType >::AdjustEdge(), keepRunning, and Markov::API::Concurrency::ThreadSharedListHa
Referenced by Markov::API::MarkovPasswords::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.11.3 Member Data Documentation

8.11.3.1 alternatingKernels

`int Markov::API::CUDA::CUDAModelMatrix::alternatingKernels [private]`
Definition at line 135 of file cudaModelMatrix.h.
Referenced by FastRandomWalk().

8.11.3.2 cudaBlocks

`int Markov::API::CUDA::CUDAModelMatrix::cudaBlocks [private]`
Definition at line 125 of file cudaModelMatrix.h.
Referenced by FastRandomWalk().

8.11.3.3 cudaGridSize

`int Markov::API::CUDA::CUDAModelMatrix::cudaGridSize [private]`
Definition at line 131 of file cudaModelMatrix.h.
Referenced by FastRandomWalk(), and prepKernelMemoryChannel().

8.11.3.4 cudaMemPerGrid

`int Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid [private]`
Definition at line 132 of file cudaModelMatrix.h.
Referenced by FastRandomWalk(), and GatherAsyncKernelOutput().

8.11.3.5 cudaPerKernelAllocationSize

`long int Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize [private]`
Definition at line 133 of file cudaModelMatrix.h.
Referenced by FastRandomWalk(), GatherAsyncKernelOutput(), and prepKernelMemoryChannel().

8.11.3.6 cudastreams

`cudaStream_t* Markov::API::CUDA::CUDAModelMatrix::cudastreams [private]`
Definition at line 139 of file cudaModelMatrix.h.

8.11.3.7 cudaThreads

`int Markov::API::CUDA::CUDAModelMatrix::cudaThreads [private]`
Definition at line 126 of file cudaModelMatrix.h.
Referenced by FastRandomWalk().

8.11.3.8 datasetFile

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

8.11.3.9 device_edgeMatrix

`char* Markov::API::CUDA::CUDAModelMatrix::device_edgeMatrix [private]`
VRAM Address pointer of edge matrix (from modelMatrix.h)
Definition at line 88 of file cudaModelMatrix.h.

8.11.3.10 device_matrixIndex

`char* Markov::API::CUDA::CUDAModelMatrix::device_matrixIndex [private]`
VRAM Address pointer of matrixIndex (from modelMatrix.h)
Definition at line 98 of file cudaModelMatrix.h.

8.11.3.11 device_outputBuffer

`char** Markov::API::CUDA::CUDAModelMatrix::device_outputBuffer [private]`
RandomWalk results in device.
Definition at line 108 of file cudaModelMatrix.h.
Referenced by prepKernelMemoryChannel().

8.11.3.12 device_seeds

`unsigned long** Markov::API::CUDA::CUDAModelMatrix::device_seeds [private]`
Definition at line 137 of file cudaModelMatrix.h.
Referenced by prepKernelMemoryChannel().

8.11.3.13 device_totalEdgeWeights

`long int* Markov::API::CUDA::CUDAModelMatrix::device_totalEdgeWeights [private]`
VRAM Address pointer of total edge weights (from modelMatrix.h)
Definition at line 103 of file cudaModelMatrix.h.

8.11.3.14 device_valueMatrix

`long int* Markov::API::CUDA::CUDAModelMatrix::device_valueMatrix [private]`
VRAM Address pointer of value matrix (from modelMatrix.h)
Definition at line 93 of file cudaModelMatrix.h.

8.11.3.15 edgeMatrix

`char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]`
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line 175 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.11.3.16 edges

`std::vector<Edge<char >*> Markov::Model< char >::edges [private], [inherited]`
A list of all edges in this model.
Definition at line 204 of file model.h.

8.11.3.17 flatEdgeMatrix

`char* Markov::API::CUDA::CUDAModelMatrix::flatEdgeMatrix [private]`
Adding Edge matrix end-to-end and resize to 1-D array for better perfomance on traversing.
Definition at line 118 of file cudaModelMatrix.h.
Referenced by FlattenMatrix().

8.11.3.18 flatValueMatrix

`long int* Markov::API::CUDA::CUDAModelMatrix::flatValueMatrix [private]`
Adding Value matrix end-to-end and resize to 1-D array for better perfomance on traversing.
Definition at line 123 of file cudaModelMatrix.h.
Referenced by FlattenMatrix().

8.11.3.19 iterationsPerKernelThread

`int Markov::API::CUDA::CUDAModelMatrix::iterationsPerKernelThread [private]`
Definition at line 127 of file cudaModelMatrix.h.
Referenced by FastRandomWalk().

8.11.3.20 matrixIndex

`char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]`
to hold the Matrix index (To hold the orders of 2-D arrays')

Definition at line 190 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.11.3.21 matrixSize

`int Markov::API::ModelMatrix::matrixSize [protected], [inherited]`
to hold Matrix size
Definition at line 185 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), Markov::API::ModelMatrix::FastRandomWalkThread(), and FlattenMatrix().

8.11.3.22 modelSavefile

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

Definition at line 124 of file markovPasswords.h.

8.11.3.23 nodes

`std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]`
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.

8.11.3.24 numberOfPartitions

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

8.11.3.25 outputBuffer

`char** Markov::API::CUDA::CUDAModelMatrix::outputBuffer [private]`
RandomWalk results in host.
Definition at line 113 of file cudaModelMatrix.h.
Referenced by GatherAsyncKernelOutput(), and prepKernelMemoryChannel().

8.11.3.26 outputFile

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

Definition at line 125 of file markovPasswords.h.

8.11.3.27 ready

`bool Markov::API::ModelMatrix::ready [protected], [inherited]`
True when matrix is constructed. False if not.
Definition at line 200 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::ModelMatrix().

8.11.3.28 starterNode

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

8.11.3.29 totalEdgeWeights

`long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]`
Array of the Total Edge Weights.
Definition at line 195 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.11.3.30 totalOutputPerKernel

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

8.11.3.31 totalOutputPerSync

`long int Markov::API::CUDA::CUDAModelMatrix::totalOutputPerSync [private]`
Definition at line 128 of file cudaModelMatrix.h.
Referenced by FastRandomWalk().

8.11.3.32 valueMatrix

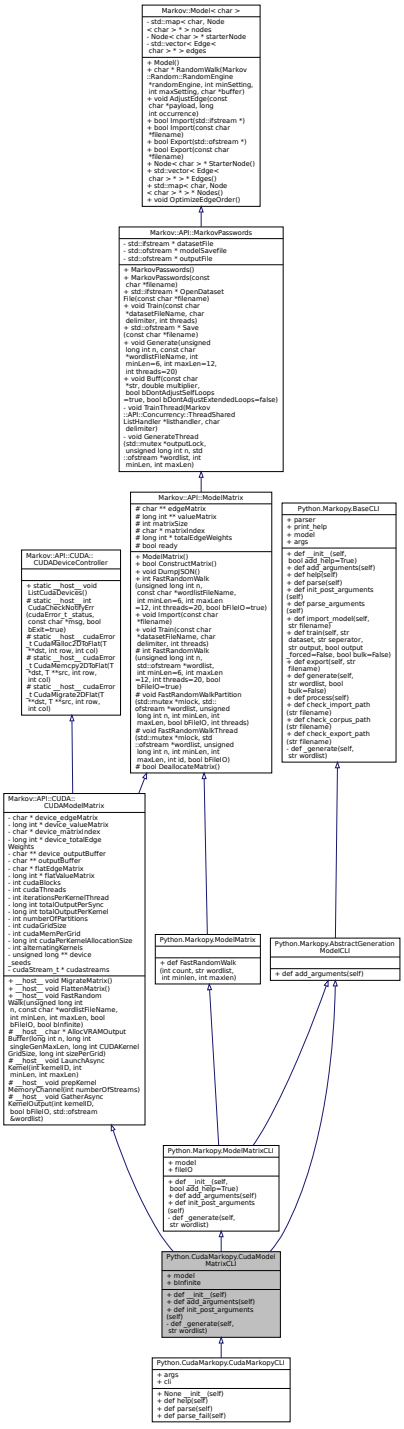
`long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]`
2-d Integer array for the value Matrix (For the weights of Edges)
Definition at line 180 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().
The documentation for this class was generated from the following files:

- Markopy/CudaMarkovAPI/src/cudaModelMatrix.h
- Markopy/CudaMarkovAPI/src/cudaModelMatrix.cu

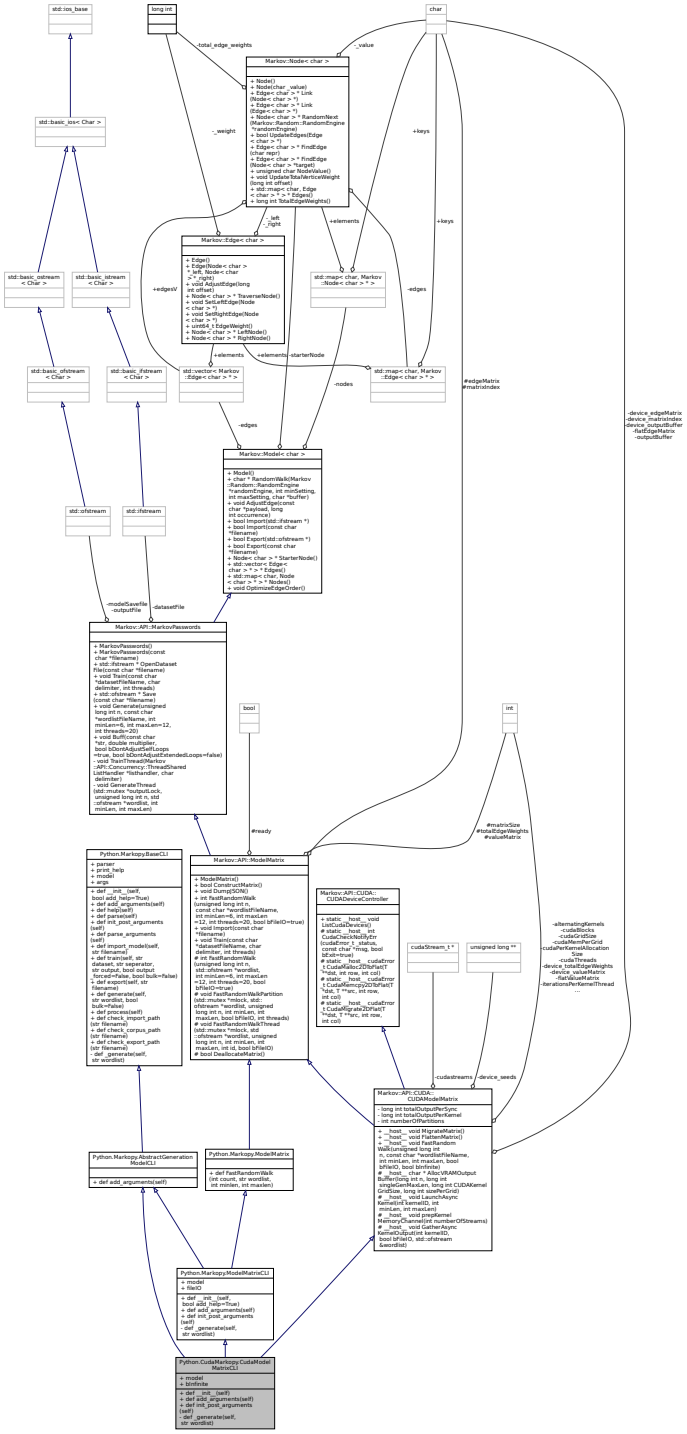
8.12 Python.CudaMarkopy.CudaModelMatrixCLI Class Reference

Python CLI wrapper for CudaModelMatrix.

Inheritance diagram for Python.CudaMarkopy.CudaModelMatrixCLI:



Collaboration diagram for Python.CudaMarkopy.CudaModelMatrixCLI:



Public Member Functions

- def __init__ (self)
- def add_arguments (self)
- def init_post_arguments (self)
- def help (self)
- def parse (self)
- def parse_arguments (self)

- def import_model (self, str filename)

  Import a model file.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def process (self)

  Process parameters for operation.
- def FastRandomWalk (int count, str wordlist, int minlen, int maxlen)
- int FastRandomWalk (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

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

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

  Debug function to dump the model to a JSON file.
- void Import (const char *filename)

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

  Import a file to construct the model.
- void Train (const char *datasetFileName, char delimiter, int threads)

  Train the model with the dataset file.
- std::ifstream * OpenDatasetFile (const char *filename)

  Open dataset file and return the ifstream pointer.
- std::ofstream * Save (const char *filename)

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

  Call Markov::Model::RandomWalk n times, and collect output.
- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- char * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

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

  Adjust the model with a single string.
- bool Export (std::ofstream *)

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

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

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

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

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.

- def help (self)
- def parse (self)
- def parse_arguments (self)
- def import_model (self, str filename)

  Import a model file.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def process (self)

  Process parameters for operation.
- __host__ void MigrateMatrix ()

  Migrate the class members to the VRAM.
- __host__ void FlattenMatrix ()

  Flatten migrated matrix from 2d to 1d.
- __host__ void FastRandomWalk (unsigned long int n, const char *wordlistFileName, int minLen, int maxLen, bool bFileIO, bool bInfinite)

  Random walk on the Matrix-reduced Markov::Model.
- int FastRandomWalk (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

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

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

  Debug function to dump the model to a JSON file.
- void Import (const char *filename)

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

  Import a file to construct the model.
- void Train (const char *datasetFileName, char delimiter, int threads)

  Train the model with the dataset file.
- std::ifstream * OpenDatasetFile (const char *filename)

  Open dataset file and return the ifstream pointer.
- std::ofstream * Save (const char *filename)

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

  Call Markov::Model::RandomWalk n times, and collect output.
- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- char * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

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

  Adjust the model with a single string.
- bool Export (std::ofstream *)

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

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

- Node< char > * StarterNode ()

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

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

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.

Static Public Member Functions

- def check_import_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- def check_import_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- static __host__ void ListCudaDevices ()

  List CUDA devices in the system.

Public Attributes

- model
- bInfinite
- fileIO
- parser
- print_help
- args
- parser
- print_help
- args

Protected Member Functions

- int FastRandomWalk (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- void FastRandomWalkPartition (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

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

  A single thread of a single partition of FastRandomWalk.
- bool DeallocateMatrix ()

  Deallocate matrix and make it ready for re-construction.

- int FastRandomWalk (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- __host__ char * AllocVRAMOutputBuffer (long int n, long int singleGenMaxLen, long int CUDAKernelGrid↩Size, long int sizePerGrid)

  Allocate the output buffer for kernel operation.
- __host__ void LaunchAsyncKernel (int kernelID, int minLen, int maxLen)
- __host__ void prepKernelMemoryChannel (int numberOfStreams)
- __host__ void GatherAsyncKernelOutput (int kernelID, bool bFileIO, std::ofstream &wordlist)
- void FastRandomWalkPartition (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

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

  A single thread of a single partition of FastRandomWalk.
- bool DeallocateMatrix ()

  Deallocate matrix and make it ready for re-construction.

Static Protected Member Functions

- static __host__ int CudaCheckNotifyErr (cudaError_t _status, const char *msg, bool bExit=true)

  Check results of the last operation on GPU.
- template<typename T >
  static __host__ cudaError_t CudaMalloc2DToFlat (T **dst, int row, int col)

  Malloc a 2D array in device space.
- template<typename T >
  static __host__ cudaError_t CudaMemcpy2DToFlat (T *dst, T **src, int row, int col)

  Memcpy a 2D array in device space after flattening.
- template<typename T >
  static __host__ cudaError_t CudaMigrate2DFlat (T **dst, T **src, int row, int col)

  Both malloc and memcpy a 2D array into device VRAM.

Protected Attributes

- char ** edgeMatrix

  2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** valueMatrix

  2-d Integer array for the value Matrix (For the weights of Edges)
- int matrixSize

  to hold Matrix size
- char * matrixIndex

  to hold the Matrix index (To hold the orders of 2-D arrays')
- long int * totalEdgeWeights

  Array of the Total Edge Weights.
- bool ready

  True when matrix is constructed. False if not.
- char ** edgeMatrix

  2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** valueMatrix

  2-d Integer array for the value Matrix (For the weights of Edges)
- int matrixSize

  to hold Matrix size
- char * matrixIndex

to hold the Matrix index (To hold the orders of 2-D arrays')

- long int * totalEdgeWeights

  Array of the Total Edge Weights.
- bool ready

  True when matrix is constructed. False if not.

Private Member Functions

- def _generate (self, str wordlist)

  wrapper for generate function.
- 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

  Dataset file input of our system
- std::ofstream * outputFile

  File to save model of our system
- std::map< char, Node< char > * > nodes

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

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

  A list of all edges in this model.
- char * device_edgeMatrix

  VRAM Address pointer of edge matrix (from modelMatrix.h)
- long int * device_valueMatrix

  VRAM Address pointer of value matrix (from modelMatrix.h)
- char * device_matrixIndex

  VRAM Address pointer of matrixIndex (from modelMatrix.h)
- long int * device_totalEdgeWeights

  VRAM Address pointer of total edge weights (from modelMatrix.h)
- char ** device_outputBuffer

  RandomWalk results in device.
- char ** outputBuffer

  RandomWalk results in host.
- char * flatEdgeMatrix

  Adding Edge matrix end-to-end and resize to 1-D array for better perfomance on traversing.
- long int * flatValueMatrix

  Adding Value matrix end-to-end and resize to 1-D array for better perfomance on traversing.
- int cudaBlocks
- int cudaThreads
- int iterationsPerKernelThread
- long int totalOutputPerSync
- long int totalOutputPerKernel
- int numberOfPartitions

- int cudaGridSize
- int cudaMemPerGrid
- long int cudaPerKernelAllocationSize
- int alternatingKernels
- unsigned long ** device_seeds
- cudaStream_t * cudastreams

8.12.1 Detailed Description

Python CLI wrapper for CudaModelMatrix.
Definition at line 55 of file cudammx.py.

8.12.2 Constructor & Destructor Documentation

8.12.2.1 __init__()

```
def Python.CudaMarkopy.CudaModelMatrixCLI.__init__ (
              self )
```

Reimplemented in Python.CudaMarkopy.CudaMarkopyCLI.
Definition at line 63 of file cudammx.py.

```
00063     def __init__(self):
00064         super().__init__()
00065         self.model = cudamarkopy.CUDAModelMatrix()
00066
```

8.12.3 Member Function Documentation

8.12.3.1 _generate()

```
def Python.CudaMarkopy.CudaModelMatrixCLI._generate (
              self,
             str  wordlist )  [private]
```

wrapper for generate function.
This can be overloaded by other models

Parameters

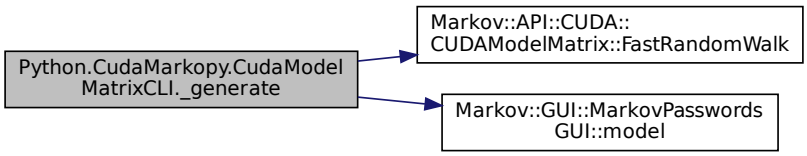
wordlist	filename to generate to

Reimplemented from Python.Markopy.ModelMatrixCLI.
Definition at line 75 of file cudammx.py.

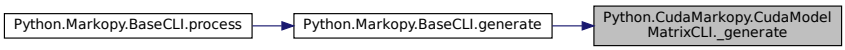
```
00075     def _generate(self, wordlist : str ):
00076         self.model.FastRandomWalk(int(self.args.count), wordlist, int(self.args.min),
      int(self.args.max), self.fileIO, self.bInfinite)
00077
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.CudaMarkopy.CudaModelMatrixCLI.bInfinite, Markov::API::CUDA::CUDAModelMatrix.FastRandomWalk(), Python.Markopy.ModelMatrixCLI.fileIO, Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.mod
Referenced by Python.Markopy.BaseCLI.generate().

Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.2 add_arguments()

```
def Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments (
              self )
```

Reimplemented from Python.Markopy.AbstractGenerationModelCLI.

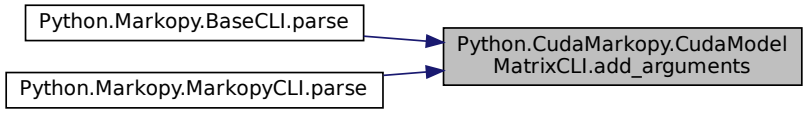
Definition at line 67 of file cudammx.py.

```
00067     def add_arguments(self):
00068         super().add_arguments()
00069         self.parser.add_argument("-if", "--infinite", action="store_true", help="Infinite generation
      mode")
00070
```

References Python.Markopy.BaseCLI.parser.

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.12.3.3 AdjustEdge() [1/2]

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

Adjust the model with a single string.

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

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

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

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

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

Parameters

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

Definition at line 109 of file model.h.

```
00337                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.12.3.4 AdjustEdge() [2/2]

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

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

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

Parameters

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

Definition at line 109 of file model.h.

```
00337                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
```

```
00355 }
```

8.12.3.5 AllocVRAMOutputBuffer()

```
__host__ char* Markov::API::CUDA::CUDAModelMatrix::AllocVRAMOutputBuffer (
            long int n,
            long int singleGenMaxLen,
            long int CUDAKernelGridSize,
            long int sizePerGrid )  [protected], [inherited]
```

Allocate the output buffer for kernel operation.
TODO

Parameters

n	- Number of passwords to generate.
singleGenMaxLen	- maximum string length for a single generation
CUDAKernelGridSize	- Total number of grid members in CUDA kernel
sizePerGrid	- Size to allocate per grid member

Returns

pointer to the allocation on VRAM

8.12.3.6 Buff() [1/2]

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file markovPasswords.cpp.

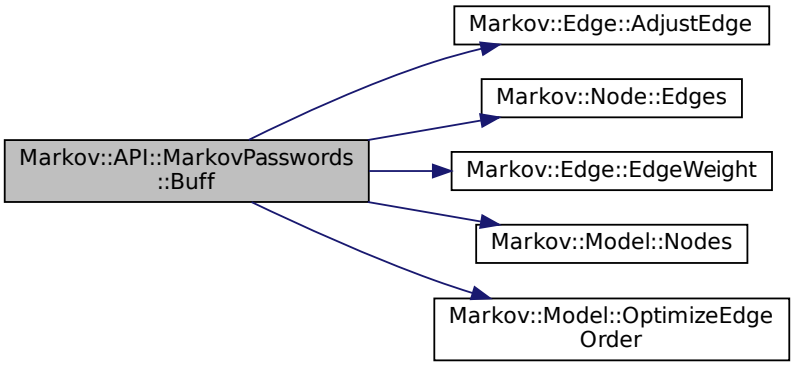
```
00153                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
```

```
00173
00174         }
00175         i++;
00176     }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

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

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.12.3.7 Buff() [2/2]

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

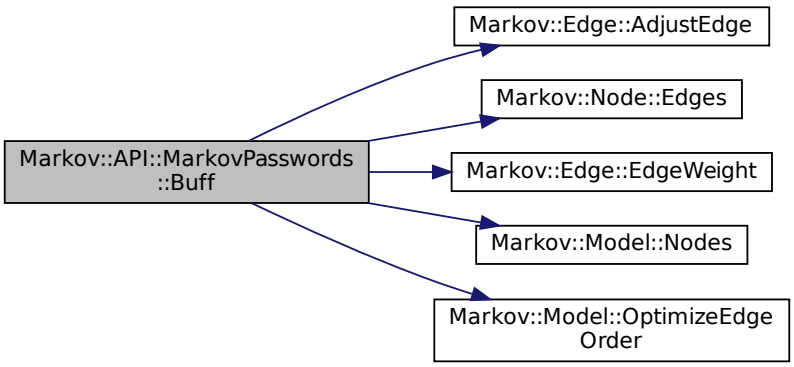
Definition at line 153 of file markovPasswords.cpp.

```
00153                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173
00174         }
00175         i++;
00176     }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

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

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.12.3.8 check_corpus_path() [1/2]

```
def Python.Markopy.BaseCLI.check_corpus_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186 
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.12.3.9 check_corpus_path() [2/2]

```
def Python.Markopy.BaseCLI.check_corpus_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186 
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.12.3.10 check_export_path() [1/2]

```
def Python.Markopy.BaseCLI.check_export_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197 
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.12.3.11 check_export_path() [2/2]

```
def Python.Markopy.BaseCLI.check_export_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197 
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.12.3.12 check_import_path() [1/2]

```
def Python.Markopy.BaseCLI.check_import_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

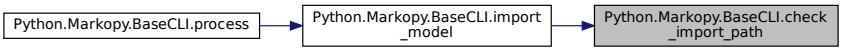
filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
```

Referenced by Python.Markopy.BaseCLI.import_model().

Here is the caller graph for this function:



8.12.3.13 check_import_path() [2/2]

```
def Python.Markopy.BaseCLI.check_import_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

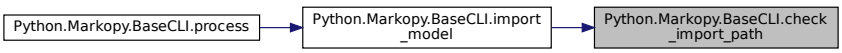
filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
```

Referenced by Python.Markopy.BaseCLI.import_model().

Here is the caller graph for this function:



8.12.3.14 ConstructMatrix() [1/2]

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

Construct the related Matrix data for the model.
This operation can be used after importing/training to allocate and populate the matrix content.
this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

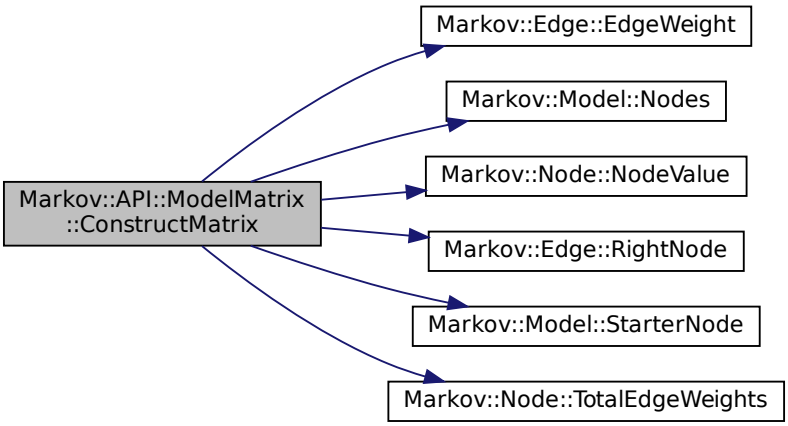
True if constructed. False if already construced.

Definition at line 31 of file modelMatrix.cpp.

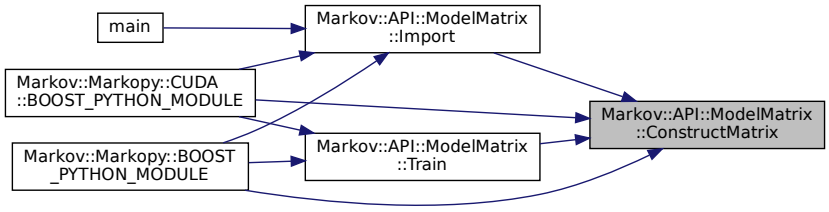
```
00031                                           {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034 
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037 
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072 
00073         }
00074         i++;
00075     }
00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::API::ModelMatrix::n
Markov::API::ModelMatrix::matrixSize, Markov::Model< NodeStorageType >::Nodes(), Markov::Node< storageType >::NodeValue()
Markov::API::ModelMatrix::ready, Markov::Edge< NodeStorageType >::RightNode(), Markov::Model< NodeStorageType >::StarterN
Markov::API::ModelMatrix::totalEdgeWeights, Markov::Node< storageType >::TotalEdgeWeights(), and Markov::API::ModelMatrix::va
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(),
Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().

Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.15 ConstructMatrix() [2/2]

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

Construct the related Matrix data for the model.
This operation can be used after importing/training to allocate and populate the matrix content.
this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

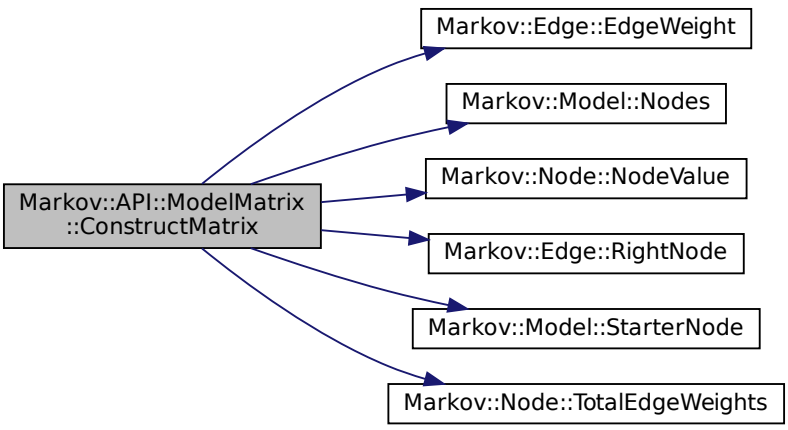
True if constructed. False if already construced.

Definition at line 31 of file modelMatrix.cpp.

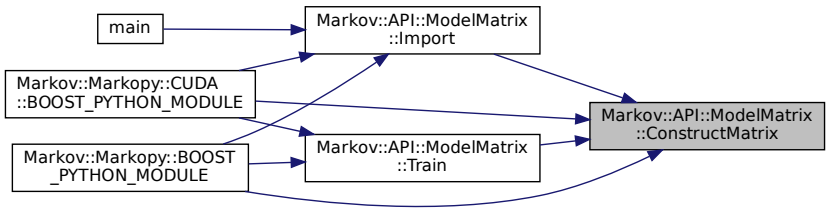
```
00031                                          {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034 
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037 
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
```

```
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072
00073         }
00074         i++;
00075     }
00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::API::ModelMatrix::r
Markov::API::ModelMatrix::matrixSize, Markov::Model< NodeStorageType >::Nodes(), Markov::Node< storageType >::NodeValue()
Markov::API::ModelMatrix::ready, Markov::Edge< NodeStorageType >::RightNode(), Markov::Model< NodeStorageType >::StarterN
Markov::API::ModelMatrix::totalEdgeWeights, Markov::Node< storageType >::TotalEdgeWeights(), and Markov::API::ModelMatrix::va
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(),
Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.16 CudaCheckNotifyErr()

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

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

Parameters

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

Returns

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

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

Definition at line 32 of file cudaDeviceController.cu.

```
00032
                    {
00033         if (_status != cudaSuccess) {
00034             std::cerr « "\033[1;31m" « msg  « " -> " « cudaGetErrorString(_status)  « " ("« _status «
      ")" « "\033[0m" « "\n";
00035
00036             if(bExit) {
00037                 cudaDeviceReset();
00038                 exit(1);
00039             }
00040         }
00041         return 0;
00042     }
```

8.12.3.17 CudaMalloc2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat (
            T ** dst,
            int row,
            int col )  [inline], [static], [protected], [inherited]
```

Malloc a 2D array in device space.
This function will allocate enough space on VRAM for flattened 2D array.

Parameters

dst	destination pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

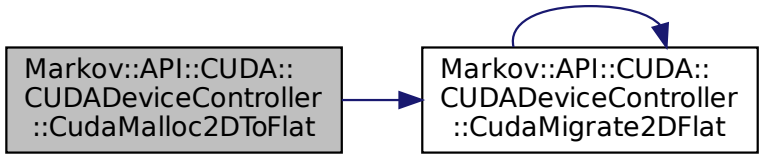
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess){
    CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
}
```

Definition at line 75 of file cudaDeviceController.h.

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

References Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().

Here is the call graph for this function:



8.12.3.18 CudaMemcpy2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat (
            T * dst,
            T ** src,
            int row,
            int col )  [inline], [static], [protected], [inherited]
```

Memcpy a 2D array in device space after flattening.

Resulting buffer will not be true 2D array.

Parameters

dst	destination pointer
rc	source pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

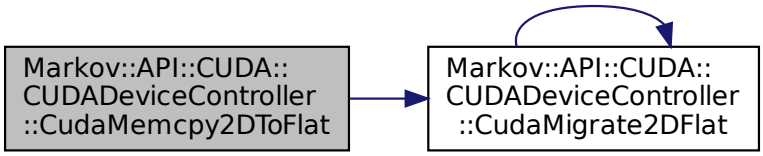
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
CudaCheckNotifyErr(cudastatus, "  CudaMemcpy2DToFlat Failed.", false);
```

Definition at line 103 of file cudaDeviceController.h.

```
00103                                                                                                {
00104             T* tempbuf = new T[row*col];
00105             for(int i=0;i<row;i++){
00106                 memcpy(&(tempbuf[row*i]), src[i], col);
00107             }
00108             return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00109
00110         }
```

References Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().

Here is the call graph for this function:



8.12.3.19 CudaMigrate2DFlat()

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

Both malloc and memcpy a 2D array into device VRAM.

Resulting buffer will not be true 2D array.

Parameters

dst	destination pointer
rc	source pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMigrate2DFlat<long int>(
   &dst, this->valueMatrix, this->matrixSize, this->matrixSize);
CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
```

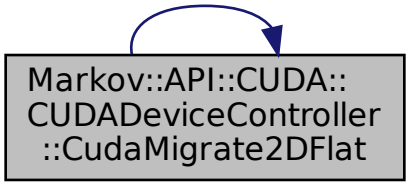
Definition at line 132 of file cudaDeviceController.h.

```
00132                                                                                                                {
00133             cudaError_t cudastatus;
00134             cudastatus = CudaMalloc2DToFlat<T>(dst, row, col);
00135             if(cudastatus!=cudaSuccess){
00136                 CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
00137                 return cudastatus;
00138             }
00139             cudastatus = CudaMemcpy2DToFlat<T>(*dst,src,row,col);
00140             CudaCheckNotifyErr(cudastatus, "  CudaMemcpy2DToFlat Failed.", false);
00141             return cudastatus;
00142         }
```

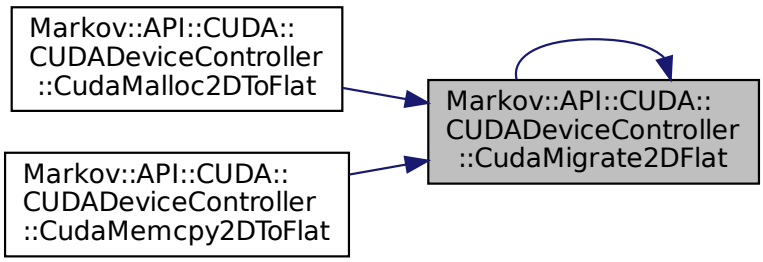
References Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().

Referenced by Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat(), Markov::API::CUDA::CUDADeviceController::Cu and Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().

Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.20 DeallocateMatrix() [1/2]

```
bool Markov::API::ModelMatrix::DeallocateMatrix ( )  [protected], [inherited]
```

Deallocate matrix and make it ready for re-construction.

Returns

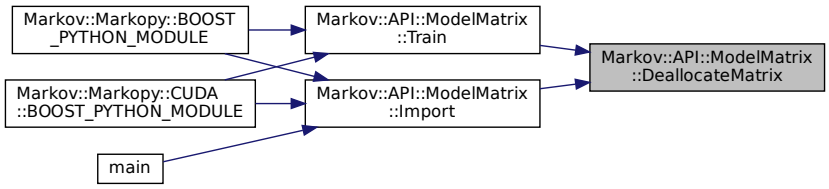
True if deallocated. False if matrix was not initialized

Definition at line 81 of file modelMatrix.cpp.

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

```
00085
00086     for(int i=0;i<this->matrixSize;i++){
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090
00091     for(int i=0;i<this->matrixSize;i++){
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
00099 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, Markov::API::ModelMatrix::ready, Markov::API::ModelMatrix::totalEdgeWeights, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().
Here is the caller graph for this function:



8.12.3.21 DeallocateMatrix() [2/2]

```
bool Markov::API::ModelMatrix::DeallocateMatrix ( )  [protected], [inherited]
```

Deallocate matrix and make it ready for re-construction.

Returns

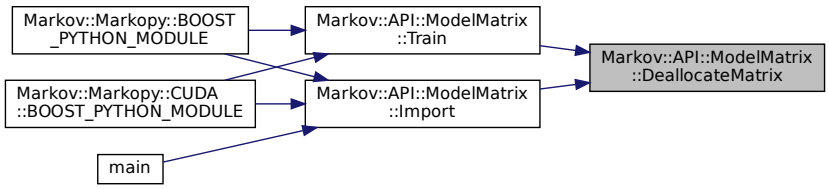
True if deallocated. False if matrix was not initialized

Definition at line 81 of file modelMatrix.cpp.

```
00081                                           {
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
00085
00086     for(int i=0;i<this->matrixSize;i++){
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090
00091     for(int i=0;i<this->matrixSize;i++){
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
00099 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, Markov::API::ModelMatrix::ready, Markov::API::ModelMatrix::totalEdgeWeights, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().

Here is the caller graph for this function:



8.12.3.22 DumpJSON() [1/2]

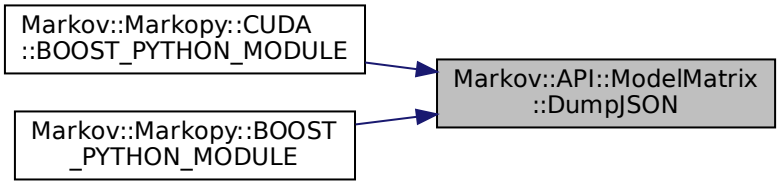
`void Markov::API::ModelMatrix::DumpJSON ( ) [inherited]`

Debug function to dump the model to a JSON file.
Might not work 100%. Not meant for production use.
Definition at line 101 of file modelMatrix.cpp.

```
00101                                   {
00102
00103     std::cout « "{\n   \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout « "\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout « "\\\\";
00107         else if(this->matrixIndex[i]==0) std::cout « "\\\\x00";
00108         else if(i==0) std::cout « "\\\\xff";
00109         else if(this->matrixIndex[i]=='\n') std::cout « "\\n";
00110         else std::cout « this->matrixIndex[i];
00111     }
00112     std::cout «
00113     "\",\n"
00114     "   \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++){
00117         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00120         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00121         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00122         for(int j=0;j<this->matrixSize;j++){
00123             if(this->edgeMatrix[i][j]=='"') std::cout « "\"\\\"\"";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout « "\"\\\\\"";
00125             else if(this->edgeMatrix[i][j]==0) std::cout « "\"\\\\x00\"";
00126             else if(this->edgeMatrix[i][j]<0) std::cout « "\"\\\\xff\"";
00127             else if(this->matrixIndex[i]=='\n') std::cout « "\"\\n\"";
00128             else std::cout « "\"" « this->edgeMatrix[i][j] « "\"";
00129             if(j!=this->matrixSize-1) std::cout « ", ";
00130         }
00131         std::cout « "],\n";
00132     }
00133     std::cout « "},\n";
00134
00135     std::cout « "\"   weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++){
00137         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00140         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00141         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++){
00144             std::cout « this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout « ", ";
00146         }
00147         std::cout « "],\n";
00148     }
00149     std::cout « "  }\n}\n";
00150 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), and Markov::Markopy::BOOST_PYTHON_MODULE().

Here is the caller graph for this function:



8.12.3.23 DumpJSON() [2/2]

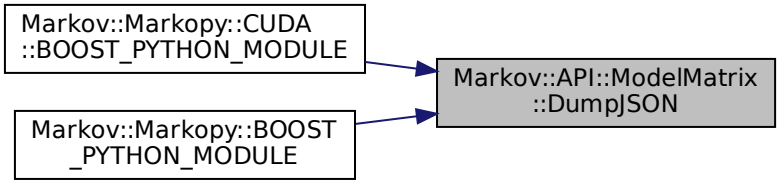
```
void Markov::API::ModelMatrix::DumpJSON ( )  [inherited]
```

Debug function to dump the model to a JSON file.
Might not work 100%. Not meant for production use.
Definition at line 101 of file modelMatrix.cpp.

```
00101                                   {
00102
00103     std::cout « "{\n   \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout « "\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout « "\\\\";
00107         else if(this->matrixIndex[i]==0) std::cout « "\\\\x00";
00108         else if(i==0) std::cout « "\\\\xff";
00109         else if(this->matrixIndex[i]=='\n') std::cout « "\\n";
00110         else std::cout « this->matrixIndex[i];
00111     }
00112     std::cout «
00113     "\",\n"
00114     "   \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++){
00117         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00120         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00121         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00122         for(int j=0;j<this->matrixSize;j++){
00123             if(this->edgeMatrix[i][j]=='"') std::cout « "\"\\\"\"";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout « "\"\\\\\"";
00125             else if(this->edgeMatrix[i][j]==0) std::cout « "\"\\\\x00\"";
00126             else if(this->edgeMatrix[i][j]<0) std::cout « "\"\\\\xff\"";
00127             else if(this->matrixIndex[i]=='\n') std::cout « "\"\\n\"";
00128             else std::cout « "\"" « this->edgeMatrix[i][j] « "\"";
00129             if(j!=this->matrixSize-1) std::cout « ", ";
00130         }
00131         std::cout « "],\n";
00132     }
00133     std::cout « "},\n";
00134
00135     std::cout « "\"   weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++){
00137         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00140         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00141         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++){
00144             std::cout « this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout « ", ";
00146         }
00147         std::cout « "],\n";
00148     }
00149     std::cout « "  }\n}\n";
00150 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), and Markov::Markopy::BOOST_PYTHON_MODULE().

Here is the caller graph for this function:



8.12.3.24 Edges() [1/2]

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

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.12.3.25 Edges() [2/2]

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

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.12.3.26 Export() [1/4]

```
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 166 of file model.h.

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

8.12.3.27 Export() [2/4]

```
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 166 of file model.h.

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

8.12.3.28 export() [1/2]

```
def Python.Markopy.BaseCLI.export (
             self,
            str filename )  [inherited]
```

Export model to a file.

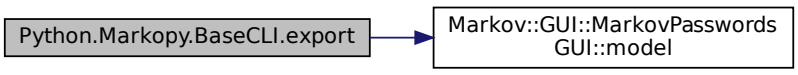
Parameters

filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.12.3.29 export() [2/2]

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

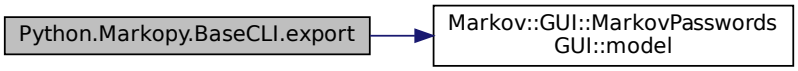
Parameters

filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.12.3.30 Export() [3/4]

```
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 155 of file model.h.

```
00288                                                {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
```

```
00292         //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
      e->RightNode()->NodeValue() « "\n";
00293         *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.12.3.31 Export() [4/4]

```
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 155 of file model.h.

```
00288                                                   {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
      e->RightNode()->NodeValue() « "\n";
00293         *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
      "\n";
00294     }
00295
00296     return true;
00297 }
```

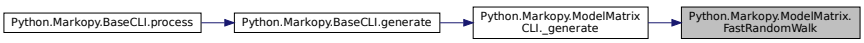
8.12.3.32 FastRandomWalk() [1/6]

```
def Python.Markopy.ModelMatrix.FastRandomWalk (
            int count,
            str wordlist,
            int minlen,
            int maxlen )  [inherited]
```

Definition at line 48 of file mm.py.

```
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass
```

Referenced by Python.Markopy.ModelMatrixCLI._generate().
Here is the caller graph for this function:



8.12.3.33 FastRandomWalk() [2/6]

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

```
            int maxLen,
            bool bFileIO,
            bool bInfinite )  [inherited]
```

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

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

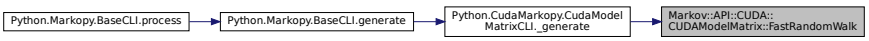
```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 58 of file cudaModelMatrix.cu.

```
00058                                                                   {
00059         cudaDeviceProp prop;
00060         int device=0;
00061         cudaGetDeviceProperties(&prop, device);
00062         cudaChooseDevice(&device, &prop);
00063         //std::cout « "Flattening matrix." « std::endl;
00064         this->FlattenMatrix();
00065         //std::cout « "Migrating matrix." « std::endl;
00066         this->MigrateMatrix();
00067         //std::cout « "Migrated matrix." « std::endl;
00068         std::ofstream wordlist;
00069         if(bFileIO)
00070             wordlist.open(wordlistFileName);
00071
00072
00073         cudaBlocks = 1024;
00074         cudaThreads = 256;
00075         iterationsPerKernelThread = 100;
00076         alternatingKernels = 2;
00077         totalOutputPerKernel = (long int)cudaBlocks*(long int)cudaThreads*iterationsPerKernelThread;
00078         totalOutputPerSync= totalOutputPerKernel*alternatingKernels;
00079         numberOfPartitions = n/totalOutputPerSync;
00080         cudaGridSize = cudaBlocks*cudaThreads;
00081         cudaMemPerGrid = (maxLen+2)*iterationsPerKernelThread;
00082         cudaPerKernelAllocationSize = cudaGridSize*cudaMemPerGrid;
00083         this->prepKernelMemoryChannel(alternatingKernels);
00084
00085         unsigned long int leftover = n - (totalOutputPerSync*numberOfPartitions);
00086
00087         if(bInfinite && !numberOfPartitions) numberOfPartitions=5;
00088         std::cerr « cudaPerKernelAllocationSize « "\n";
00089
00090         if(n%totalOutputPerSync) std::cerr « "For optimization, request outputs muliples of "«
      totalOutputPerSync « ".\n";
00091
00092         //start kernelID 1
00093         this->LaunchAsyncKernel(1, minLen, maxLen);
00094
00095         for(int i=1;i<numberOfPartitions;i++){
00096             if(bInfinite) i=0;
00097
00098             //wait kernelID1 to finish, and start kernelID 0
00099             cudaStreamSynchronize(this->cudastreams[1]);
00100             this->LaunchAsyncKernel(0, minLen, maxLen);
00101
00102             //start memcpy from kernel 1 (block until done)
00103             this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00104
00105             //wait kernelID 0 to finish, then start kernelID1
00106             cudaStreamSynchronize(this->cudastreams[0]);
00107             this->LaunchAsyncKernel(1, minLen, maxLen);
00108
00109             //start memcpy from kernel 0 (block until done)
00110             this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00111
00112         }
00113
00114         //wait kernelID1 to finish, and start kernelID 0
```

```
00115         cudaStreamSynchronize(this->cudastreams[1]);
00116         this->LaunchAsyncKernel(0, minLen, maxLen);
00117         this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00118         cudaStreamSynchronize(this->cudastreams[0]);
00119         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00120 
00121 
00122         if(!leftover) return;
00123         alternatingKernels=1;
00124         std::cerr << "Remaining line count (" << leftover << ") is lower than partition. Adjusting CUDA
      workload..\n";
00125         this->iterationsPerKernelThread = leftover/cudaGridSize;
00126         this->LaunchAsyncKernel(0, minLen, maxLen);
00127         cudaStreamSynchronize(this->cudastreams[0]);
00128         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00129 
00130         leftover -= this->iterationsPerKernelThread*cudaGridSize;
00131         if(!leftover) return;
00132 
00133         std::cerr << "Remaining line count (" << leftover << ") is lower than minimum possible. Handing
      over to CPU generation.\n";
00134         this->iterationsPerKernelThread = leftover/cudaGridSize;
00135 
00136         leftover -= this->iterationsPerKernelThread;
00137 
00138         if(!leftover) return;
00139         std::cerr << "Remaining " << leftover << " lines are absolutely not worth printing.\n";
00140         Markov::API::ModelMatrix::ConstructMatrix();
00141         Markov::API::ModelMatrix::FastRandomWalk(leftover, &wordlist, minLen, maxLen, 1, bFileIO);
00142 
00143     }
```

References Markov::API::CUDA::CUDAModelMatrix::alternatingKernels, Markov::API::CUDA::CUDAModelMatrix::cudaBlocks, Markov::API::CUDA::CUDAModelMatrix::cudaGridSize, Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid, Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize, Markov::API::CUDA::CUDAModelMatrix::cudaThreads, Markov::API::CUDA::CUDAModelMatrix::iterationsPerKernelThread, Markov::API::CUDA::CUDAModelMatrix::numberOfPartitions, Markov::API::CUDA::CUDAModelMatrix::totalOutputPerKernel, and Markov::API::CUDA::CUDAModelMatrix::totalOutputPerSync.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI::_generate().
Here is the caller graph for this function:



8.12.3.34 FastRandomWalk() [3/6]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [inherited]
```

Random walk on the Matrix-reduced Markov::Model.
This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.
While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to

Parameters

minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

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

Definition at line 217 of file modelMatrix.cpp.

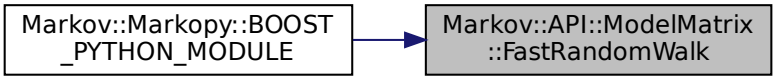
```
00217                                                {
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

References Markov::API::ModelMatrix::FastRandomWalk().

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

Here is the call graph for this function:

Here is the caller graph for this function:



8.12.3.35 FastRandomWalk() [4/6]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [inherited]
```

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

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5

This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

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

Definition at line 217 of file modelMatrix.cpp.

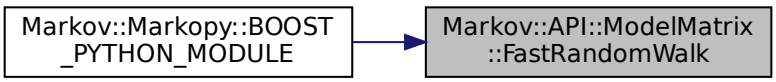
```
00217                                           {
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

References Markov::API::ModelMatrix::FastRandomWalk().

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

Here is the call graph for this function:

Here is the caller graph for this function:



8.12.3.36 FastRandomWalk() [5/6]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [protected], [inherited]
```

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

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5

This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

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

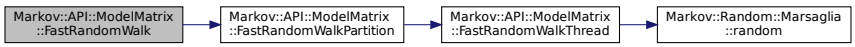
Definition at line 204 of file modelMatrix.cpp.

```
00204
                                    {
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
      threads);
00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
      threads);
00213     }
00214     return 0;
00215 }
```

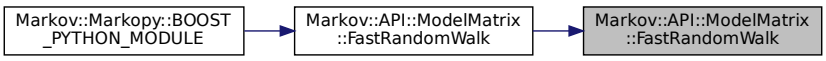
References Markov::API::ModelMatrix::FastRandomWalkPartition().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.37 FastRandomWalk() [6/6]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [protected], [inherited]
```

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

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

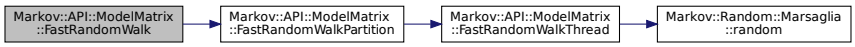
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

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

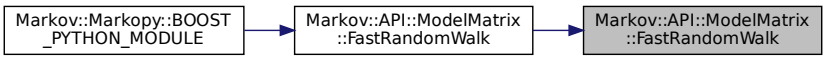
Definition at line 204 of file modelMatrix.cpp.

```
00204
                                    {
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
      threads);
00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
      threads);
00213     }
00214     return 0;
00215 }
```

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



Here is the caller graph for this function:



8.12.3.38 FastRandomWalkPartition() [1/2]

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

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

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

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

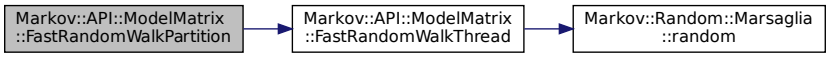
Parameters

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

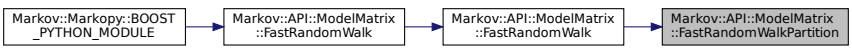
Definition at line 225 of file modelMatrix.cpp.

```
00225                                                                    {
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++){
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
      mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
      wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00239
00240     for(int i=0;i<threads;i++){
00241         threadsV[i]->join();
00242     }
00243 }
```

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



Here is the caller graph for this function:



8.12.3.39 FastRandomWalkPartition() [2/2]

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

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

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

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

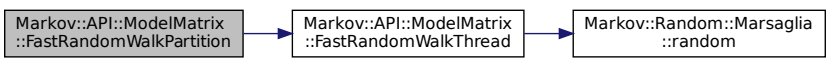
Parameters

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

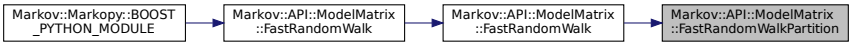
Definition at line 225 of file modelMatrix.cpp.

```
00225                                                                   {
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++){
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
       mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
       wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00239
00240     for(int i=0;i<threads;i++){
00241         threadsV[i]->join();
00242     }
00243 }
```

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



Here is the caller graph for this function:



8.12.3.40 FastRandomWalkThread() [1/2]

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

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

Parameters

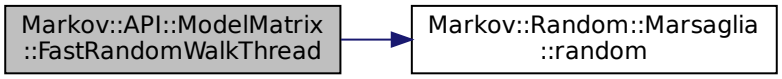
mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
id	- **DEPRECATED** Thread id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

Definition at line 153 of file modelMatrix.cpp.

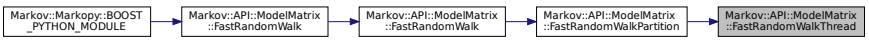
```
00153                                                                {
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179
00180             if (len >= maxLen)  break;
00181             else if ((next < 0) && (len < minLen)) continue;
00182             else if (next < 0) break;
00183             cur = next;
```

```
00184             res[bufferctr + len++] = cur;
00185         }
00186         res[bufferctr + len++] = '\n';
00187         bufferctr+=len;
00188 
00189     }
00190     if(bFileIO){
00191         mlock->lock();
00192         *wordlist « res;
00193         mlock->unlock();
00194     }else{
00195         mlock->lock();
00196         std::cout « res;
00197         mlock->unlock();
00198     }
00199     delete res;
00200 
00201 }
```

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



Here is the caller graph for this function:



8.12.3.41 FastRandomWalkThread() [2/2]

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

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

Parameters

mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

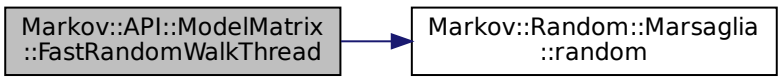
Parameters

id	- **DEPRECATED** Thread id - No longer used
bFileIO	- If false, filename will be ignored and will output to stdout.

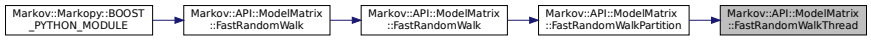
Definition at line 153 of file modelMatrix.cpp.

```
00153                                                           {
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179
00180             if (len >= maxLen)  break;
00181             else if ((next < 0) && (len < minLen)) continue;
00182             else if (next < 0) break;
00183             cur = next;
00184             res[bufferctr + len++] = cur;
00185         }
00186         res[bufferctr + len++] = '\n';
00187         bufferctr+=len;
00188
00189     }
00190     if(bFileIO){
00191         mlock->lock();
00192         *wordlist << res;
00193         mlock->unlock();
00194     }else{
00195         mlock->lock();
00196         std::cout << res;
00197         mlock->unlock();
00198     }
00199     delete res;
00200
00201 }
```

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



Here is the caller graph for this function:



8.12.3.42 FlattenMatrix()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix ( )  [inherited]
```

Flatten migrated matrix from 2d to 1d.

Definition at line 261 of file cudaModelMatrix.cu.

```
00261                                                       {
00262         this->flatEdgeMatrix = new char[this->matrixSize*this->matrixSize];
00263 
00264         this->flatValueMatrix = new long int[this->matrixSize*this->matrixSize];
00265         for(int i=0;i<this->matrixSize;i++){
00266             memcpy(&this->flatEdgeMatrix[i*this->matrixSize], this->edgeMatrix[i], this->matrixSize );
00267             memcpy(&this->flatValueMatrix[i*this->matrixSize], this->valueMatrix[i],
      this->matrixSize*sizeof(long int) );
00268         }
00269     }
```

References Markov::API::CUDA::CUDAModelMatrix::flatEdgeMatrix, Markov::API::CUDA::CUDAModelMatrix::flatValueMatrix, and Markov::API::ModelMatrix::matrixSize.

8.12.3.43 GatherAsyncKernelOutput()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput (
             int kernelID,
             bool bFileIO,
             std::ofstream & wordlist )  [protected], [inherited]
```

Definition at line 180 of file cudaModelMatrix.cu.

```
00180                                                       {
00181 
      cudaMemcpy(this->outputBuffer[kernelID],this->device_outputBuffer[kernelID],cudaPerKernelAllocationSize,
      cudaMemcpyDeviceToHost);
00182         //std::cerr << "Kernel" << kernelID << " output copied\n";
00183         if(bFileIO){
00184             for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00185                 wordlist << &this->outputBuffer[kernelID][j];
00186             }
00187         }else{
00188             for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00189                 std::cout << &this->outputBuffer[kernelID][j];
00190             }
00191         }
00192     }
```

References Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid, Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAlloc and Markov::API::CUDA::CUDAModelMatrix::outputBuffer.

8.12.3.44 generate() [1/2]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

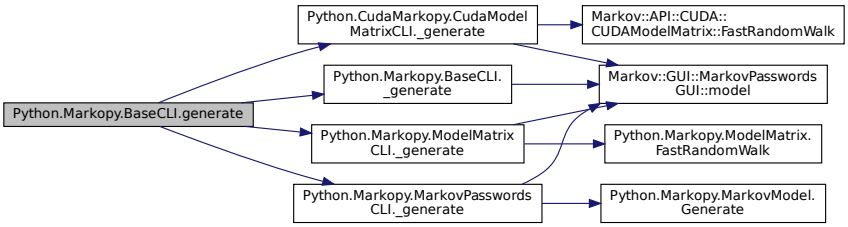
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159 
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.12.3.45 generate() [2/2]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

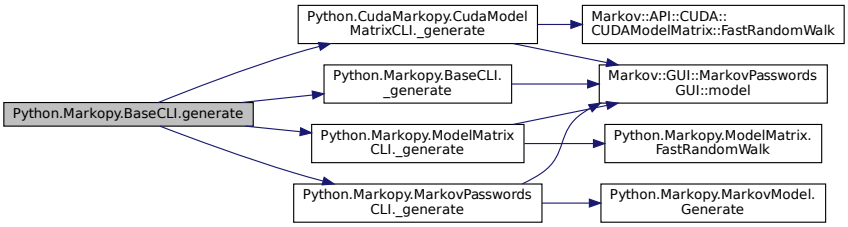
model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
```

```
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159 
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:

8.12.3.46 Generate() [1/2]

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

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

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

Parameters

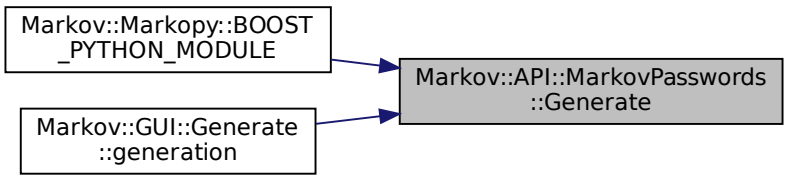
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

Definition at line 118 of file markovPasswords.cpp.

```
00118                                {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
       &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

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

Here is the caller graph for this function:



8.12.3.47 Generate() [2/2]

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

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

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

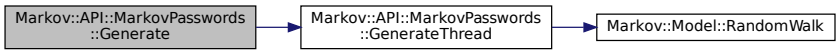
Parameters

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

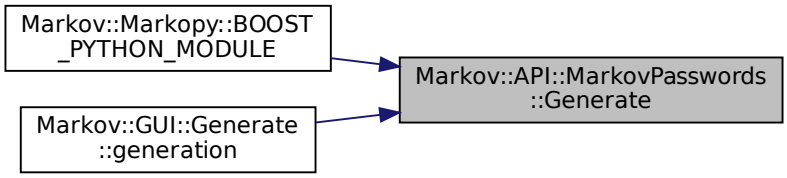
Definition at line 118 of file markovPasswords.cpp.

```
00118                                {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
      &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

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



Here is the caller graph for this function:



8.12.3.48 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
            std::mutex * outputLock,
            unsigned long int n,
            std::ofstream * wordlist,
```

```
            int minLen,
            int maxLen )  [private], [inherited]
```

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

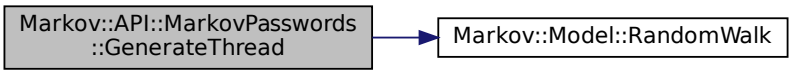
Parameters

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

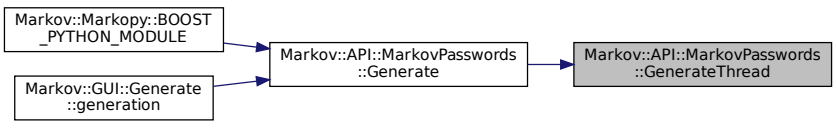
Definition at line 140 of file markovPasswords.cpp.

```
00140                                       {
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143 
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist « res « "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

References Markov::Model< NodeStorageType >::RandomWalk().
Referenced by Markov::API::MarkovPasswords::Generate().
Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.49 help() [1/2]

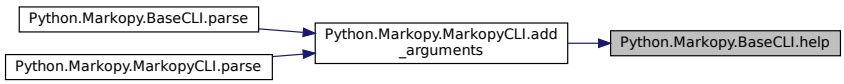
```
def Python.Markopy.BaseCLI.help (
              self )  [inherited]
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.
Definition at line 51 of file base.py.

```
00051     def help(self):
00052         "! @brief Handle help strings. Defaults to argparse's help"
```

```
00053         self.print_help()
00054
```

References Python.Markopy.BaseCLI.print_help.
Referenced by Python.Markopy.MarkopyCLI.add_arguments().
Here is the caller graph for this function:



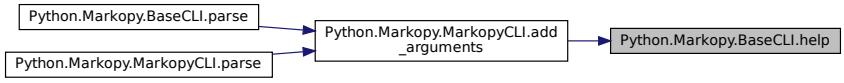
8.12.3.50 help() [2/2]

```
def Python.Markopy.BaseCLI.help (
              self )  [inherited]
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.
Definition at line 51 of file base.py.

```
00051     def help(self):
00052         "! @brief Handle help strings. Defaults to argparse's help"
00053         self.print_help()
00054
```

References Python.Markopy.BaseCLI.print_help.
Referenced by Python.Markopy.MarkopyCLI.add_arguments().
Here is the caller graph for this function:



8.12.3.51 Import() [1/4]

```
void Markov::API::ModelMatrix::Import (
             const char * filename )  [inherited]
```

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

Returns

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

Example Use: Import a file with filename

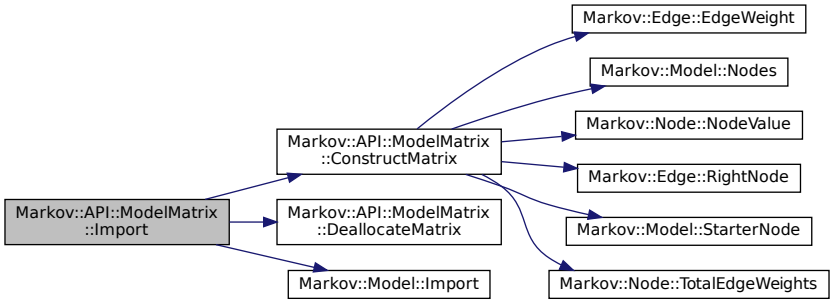
```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.
Definition at line 19 of file modelMatrix.cpp.

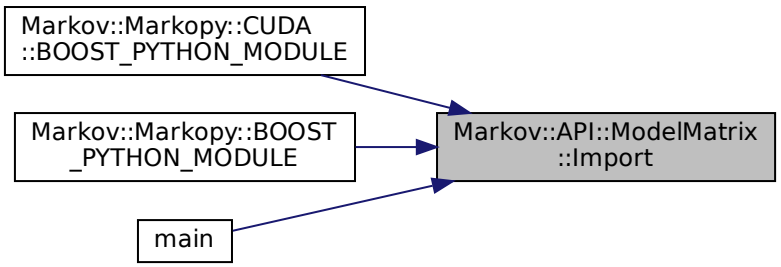
```
00019                                                 {
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::Model< NodeStorageType >::Import().
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.52 Import() [2/4]

```
void Markov::API::ModelMatrix::Import (
            const char * filename )  [inherited]
```

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

Returns

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

Example Use: Import a file with filename

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

Construct the matrix when done.

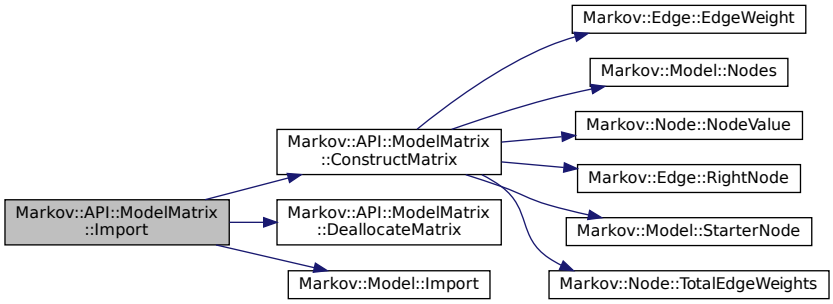
Definition at line 19 of file modelMatrix.cpp.

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

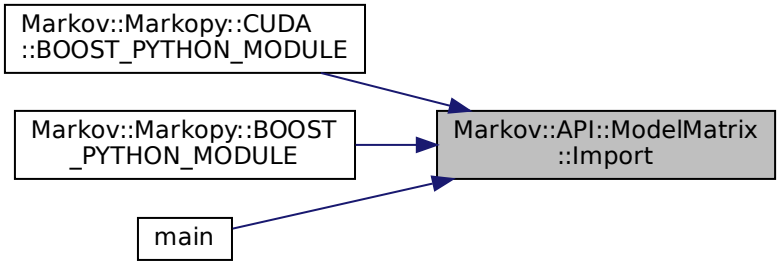
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::Model< NodeStorageType >::Import().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.53 Import() [3/4]

```
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 126 of file model.h.

```
00216                                                    {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout « "cell: " « cell « std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
```

```
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout << "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
      int(targetN->NodeValue()) << "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

8.12.3.54 Import() [4/4]

```
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 126 of file model.h.

```
00216                                                   {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
```

```
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout « "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

8.12.3.55 import_model() [1/2]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

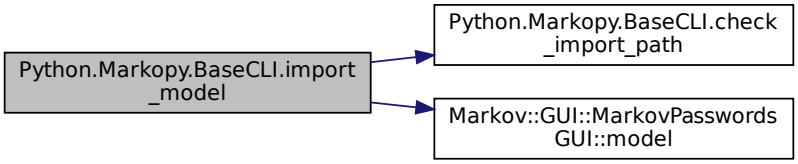
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

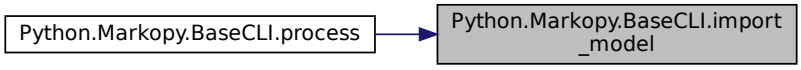
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.56 import_model() [2/2]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

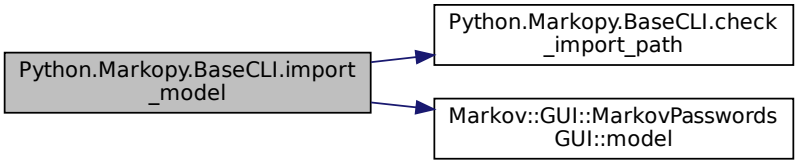
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

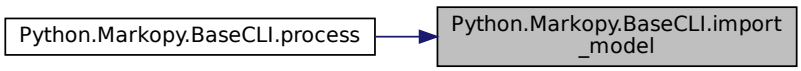
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.57 init_post_arguments()

```
def Python.CudaMarkopy.CudaModelMatrixCLI.init_post_arguments (
              self )
```

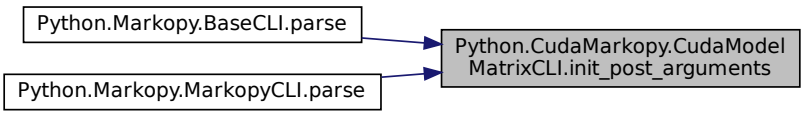
Reimplemented from Python.Markopy.ModelMatrixCLI.

Definition at line 71 of file cudammx.py.

```
00071     def init_post_arguments(self):
00072         super().init_post_arguments()
00073         self.bInfinite = self.args.infinite
00074
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.12.3.58 LaunchAsyncKernel()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::LaunchAsyncKernel (
             int kernelID,
             int minLen,
             int maxLen )  [protected], [inherited]
```

Definition at line 171 of file cudaModelMatrix.cu.

```
00171                                                                                                                {
00172
00173         //if(kernelID == 0);// cudaStreamSynchronize(this->cudastreams[2]);
00174         //else cudaStreamSynchronize(this->cudastreams[kernelID-1]);
00175         FastRandomWalkCUDAKernel<<cudaBlocks,cudaThreads,0,
      this->cudastreams[kernelID]>>(iterationsPerKernelThread, minLen, maxLen,
      this->device_outputBuffer[kernelID], this->device_matrixIndex,
00176             this->device_totalEdgeWeights, this->device_valueMatrix, this->device_edgeMatrix,
      this->matrixSize, cudaMemPerGrid, this->device_seeds[kernelID]);
00177         //std::cerr << "Started kernel" << kernelID << "\n";
00178     }
```

8.12.3.59 ListCudaDevices()

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

List CUDA devices in the system.

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

Example output:

```
Device Number: 0
Device name: GeForce RTX 2070
```

```
Memory Clock Rate (KHz): 7001000
Memory Bus Width (bits): 256
Peak Memory Bandwidth (GB/s): 448.064
Max Linear Threads: 1024
```

Definition at line 16 of file cudaDeviceController.cu.

```
00016                                                                   { //list cuda Capable devices on
       host.
00017         int nDevices;
00018         cudaGetDeviceCount(&nDevices);
00019         for (int i = 0; i < nDevices; i++) {
00020             cudaDeviceProp prop;
00021             cudaGetDeviceProperties(&prop, i);
00022             std::cerr « "Device Number: " «  i  « "\n";
00023             std::cerr « "Device name: " « prop.name « "\n";
00024             std::cerr « "Memory Clock Rate (KHz): " « prop.memoryClockRate « "\n";
00025             std::cerr « "Memory Bus Width (bits): " « prop.memoryBusWidth « "\n";
00026             std::cerr « "Peak Memory Bandwidth (GB/s): " « 2.0 * prop.memoryClockRate *
       (prop.memoryBusWidth / 8) / 1.0e6 « "\n";
00027             std::cerr « "Max Linear Threads: " « prop.maxThreadsDim[0] « "\n";
00028
00029         }
00030     }
```

8.12.3.60 MigrateMatrix()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::MigrateMatrix ( )  [inherited]
```

Migrate the class members to the VRAM.

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

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

Definition at line 20 of file cudaModelMatrix.cu.

```
00020                                                       {
00021         cudaError_t cudastatus;
00022
00023         cudastatus = cudaMalloc((char**)&(this->device_matrixIndex),
00024             this->matrixSize*sizeof(char));
00025         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_matrixIndex.");
00026
00027         cudastatus = cudaMalloc((long int **)&(this->device_totalEdgeWeights),
       this->matrixSize*sizeof(long int));
00028         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_totalEdgeWeights.");
00029
00030         cudastatus = cudaMemcpy(this->device_matrixIndex, this->matrixIndex,
00031             this->matrixSize*sizeof(char), cudaMemcpyHostToDevice);
00032         CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Index)");
00033
00034         cudastatus = cudaMemcpy(this->device_totalEdgeWeights, this->totalEdgeWeights,
00035             this->matrixSize*sizeof(long int), cudaMemcpyHostToDevice);
00036         CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Total Edge Values)");
00037
00038         cudastatus = CudaMigrate2DFlat<char>(
00039             &(this->device_edgeMatrix), this->edgeMatrix, this->matrixSize, this->matrixSize);
00040         CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize edge matrix.");
00041
00042         cudastatus = CudaMigrate2DFlat<long int>(
00043             &(this->device_valueMatrix), this->valueMatrix, this->matrixSize, this->matrixSize);
00044         CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
00045
00046     }
```

8.12.3.61 Nodes() [1/2]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.12.3.62 Nodes() [2/2]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.12.3.63 OpenDatasetFile() [1/2]

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

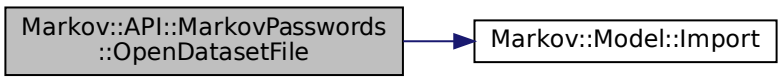
ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

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

Here is the call graph for this function:



8.12.3.64 OpenDatasetFile() [2/2]

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

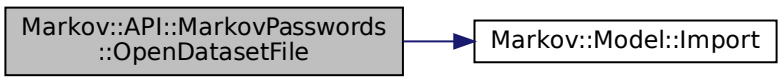
ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

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

Here is the call graph for this function:



8.12.3.65 OptimizeEdgeOrder() [1/2]

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

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.12.3.66 OptimizeEdgeOrder() [2/2]

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

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.12.3.67 parse() [1/2]

```
def Python.Markopy.BaseCLI.parse (
              self )  [inherited]
```

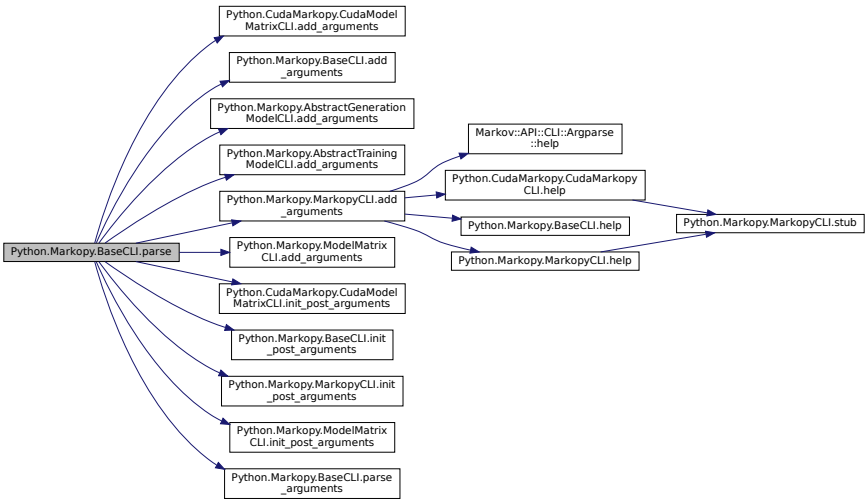
Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.

Definition at line 55 of file base.py.

```
00055     def parse(self):
00056         "! @brief add, parse and hook arguements"
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMode Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.init_post_arguments(), Python.Markopy.ModelMatrixC and Python.Markopy.BaseCLI.parse_arguments().

Here is the call graph for this function:



8.12.3.68 parse() [2/2]

```
def Python.Markopy.BaseCLI.parse (
              self )  [inherited]
```

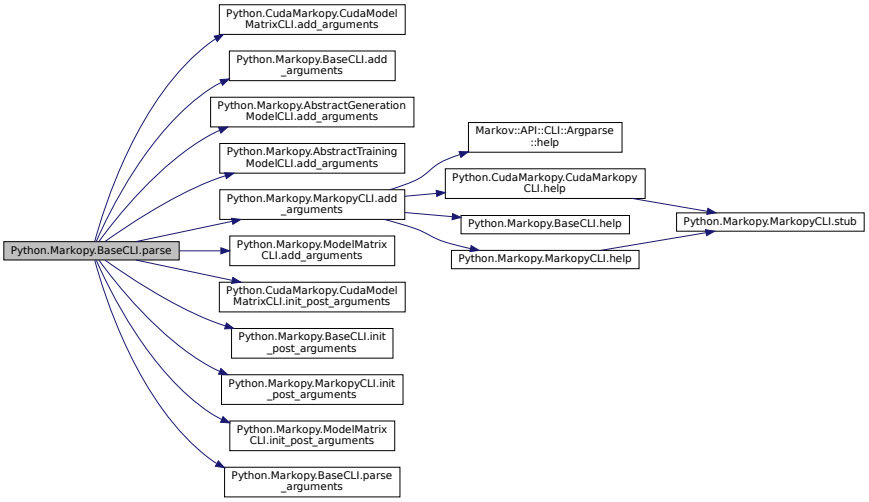
Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.

Definition at line 55 of file base.py.

```
00055     def parse(self):
00056         "! @brief add, parse and hook arguements"
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMode Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.init_post_arguments(), Python.Markopy.ModelMatrixC and Python.Markopy.BaseCLI.parse_arguments().

Here is the call graph for this function:



8.12.3.69 parse_arguments() [1/2]

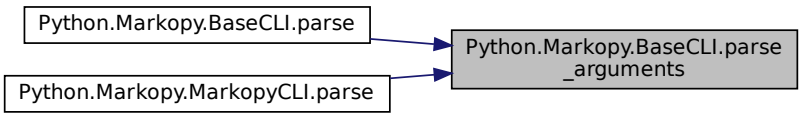
```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.12.3.70 parse_arguments() [2/2]

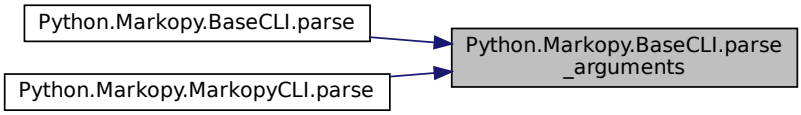
```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.12.3.71 prepKernelMemoryChannel()

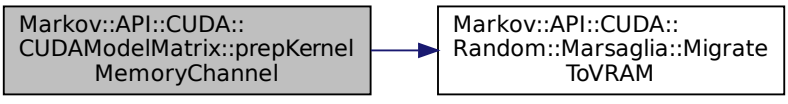
```
__host__ void Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel (
            int numberOfStreams )  [protected], [inherited]
```

Definition at line 145 of file cudaModelMatrix.cu.

```
00145                                                                                                {
00146
00147         this->cudastreams = new cudaStream_t[numberOfStreams];
00148         for(int i=0;i<numberOfStreams;i++)
00149             cudaStreamCreate(&this->cudastreams[i]);
00150
00151         this-> outputBuffer = new char*[numberOfStreams];
00152         for(int i=0;i<numberOfStreams;i++)
00153             this->outputBuffer[i]= new char[cudaPerKernelAllocationSize];
00154
00155         cudaError_t cudastatus;
00156         this-> device_outputBuffer = new char*[numberOfStreams];
00157             for(int i=0;i<numberOfStreams;i++){
00158                 cudastatus = cudaMalloc((char**)&(device_outputBuffer[i]),
      cudaPerKernelAllocationSize);
00159                 CudaCheckNotifyErr(cudastatus, "Failed to establish memory channel. Possibly out of
      VRAM?");
00160             }
00161
00162         this-> device_seeds = new unsigned long*[numberOfStreams];
00163         for(int i=0;i<numberOfStreams;i++){
00164             Markov::API::CUDA::Random::Marsaglia *MEarr = new
      Markov::API::CUDA::Random::Marsaglia[cudaGridSize];
00165             this->device_seeds[i] = Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(MEarr,
      cudaGridSize);
00166             delete[] MEarr;
00167         }
00168
00169     }
```

References Markov::API::CUDA::CUDAModelMatrix::cudaGridSize, Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocatio
Markov::API::CUDA::CUDAModelMatrix::device_outputBuffer, Markov::API::CUDA::CUDAModelMatrix::device_seeds,
Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(), and Markov::API::CUDA::CUDAModelMatrix::outputBuffer.

Here is the call graph for this function:



8.12.3.72 process() [1/2]

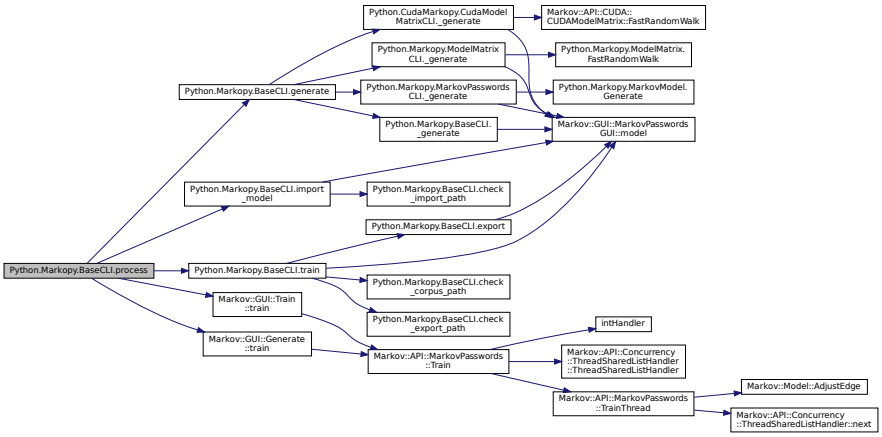
```
def Python.Markopy.BaseCLI.process (
              self )  [inherited]
```

Process parameters for operation.
Reimplemented in Python.Markopy.MarkopyCLI.
Definition at line 202 of file base.py.

```
00202     def process(self):
00203         """!
00204         @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging.pprint(f"Bulk mode operation chosen.", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
      (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00210                     corpus_list = os.listdir(self.args.dataset)
00211                     for corpus in corpus_list:
00212                         self.import_model(self.args.input)
00213                         logging.pprint(f"Training {self.args.input} with {corpus}", 2)
00214                         output_file_name = corpus
00215                         model_extension = ""
00216                         if "." in self.args.input:
00217                             model_extension = self.args.input.split(".")[-1]
00218                         self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
      f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00219                 else:
00220                     logging.pprint("In bulk training, output and dataset should be a directory.")
00221                     exit(1)
00222
00223             elif (self.args.mode.lower() == "generate"):
00224                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
      (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00225                     model_list = os.listdir(self.args.input)
00226                     print(model_list)
00227                     for input in model_list:
00228                         logging.pprint(f"Generating from {self.args.input}/{input} to
      {self.args.wordlist}/{input}.txt", 2)
00229                         self.import_model(f"{self.args.input}/{input}")
00230                         model_base = input
00231                         if "." in self.args.input:
00232                             model_base = input.split(".")[1]
00233                         self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00234                 else:
00235                     logging.pprint("In bulk generation, input and wordlist should be directory.")
00236
00237         else:
00238             self.import_model(self.args.input)
00239             if (self.args.mode.lower() == "generate"):
00240                 self.generate(self.args.wordlist)
00241
00242
00243             elif (self.args.mode.lower() == "train"):
00244                 self.train(self.args.dataset, self.args.seperator, self.args.output,
      output_forced=True)
00245
00246
00247             elif(self.args.mode.lower() == "combine"):
00248                 self.train(self.args.dataset, self.args.seperator, self.args.output)
00249                 self.generate(self.args.wordlist)
00250
00251
00252             else:
00253                 logging.pprint("Invalid mode arguement given.")
00254                 logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255                 exit(5)
00256
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.generate(), Python.Markopy.BaseCLI.import_model(), Markov::GUI::Generate.train(), Markov::GUI::Train.train(), and Python.Markopy.BaseCLI.train().

Here is the call graph for this function:



8.12.3.73 process() [2/2]

```
def Python.Markopy.BaseCLI.process (
            self )  [inherited]
```

Process parameters for operation.

Reimplemented in Python.Markopy.MarkopyCLI.

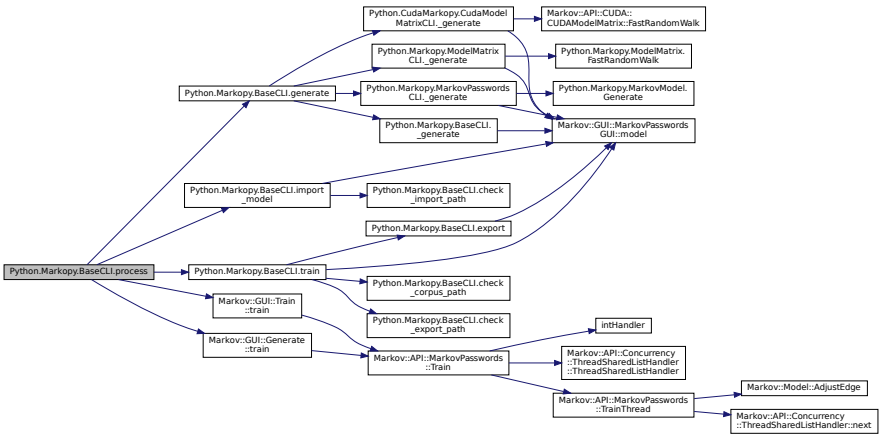
Definition at line 202 of file base.py.

```
00202     def process(self):
00203         """!
00204         @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging.pprint(f"Bulk mode operation chosen.", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
      (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00210                     corpus_list = os.listdir(self.args.dataset)
00211                     for corpus in corpus_list:
00212                         self.import_model(self.args.input)
00213                         logging.pprint(f"Training {self.args.input} with {corpus}", 2)
00214                         output_file_name = corpus
00215                         model_extension = ""
00216                         if "." in self.args.input:
00217                             model_extension = self.args.input.split(".")[-1]
00218                         self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
      f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00219                 else:
00220                     logging.pprint("In bulk training, output and dataset should be a directory.")
00221                     exit(1)
00222
00223             elif (self.args.mode.lower() == "generate"):
00224                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
      (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00225                     model_list = os.listdir(self.args.input)
00226                     print(model_list)
00227                     for input in model_list:
00228                         logging.pprint(f"Generating from {self.args.input}/{input} to
      {self.args.wordlist}/{input}.txt", 2)
00229                         self.import_model(f"{self.args.input}/{input}")
00230                         model_base = input
00231                         if "." in self.args.input:
00232                             model_base = input.split(".")[1]
00233                         self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00234                 else:
00235                     logging.pprint("In bulk generation, input and wordlist should be directory.")
00236
00237         else:
00238             self.import_model(self.args.input)
00239             if (self.args.mode.lower() == "generate"):
00240                 self.generate(self.args.wordlist)
00241
00242
00243             elif (self.args.mode.lower() == "train"):
```

```
00244                 self.train(self.args.dataset, self.args.seperator, self.args.output,
      output_forced=True)
00245
00246
00247             elif(self.args.mode.lower() == "combine"):
00248                 self.train(self.args.dataset, self.args.seperator, self.args.output)
00249                 self.generate(self.args.wordlist)
00250
00251
00252             else:
00253                 logging.pprint("Invalid mode arguement given.")
00254                 logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255                 exit(5)
00256
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.generate(), Python.Markopy.BaseCLI.import_model(), Markov::GUI::Generate.train(), Markov::GUI::Train.train(), and Python.Markopy.BaseCLI.train().

Here is the call graph for this function:



8.12.3.74 RandomWalk() [1/2]

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307
                                                        {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue();
00327     }
00328
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.12.3.75 RandomWalk() [2/2]

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

Do a random walk on this model.

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

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

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

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

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

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

```
}
```

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307                                                       {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue();
00327     }
00328
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.12.3.76 Save() [1/2]

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

Export model to file.

Parameters

filename	- Export filename.

Returns

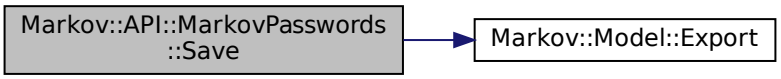
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

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

Here is the call graph for this function:



8.12.3.77 Save() [2/2]

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

Export model to file.

Parameters

filename	- Export filename.

Returns

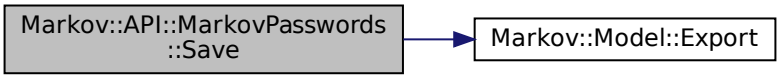
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

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

Here is the call graph for this function:



8.12.3.78 StarterNode() [1/2]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.12.3.79 StarterNode() [2/2]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.12.3.80 Train() [1/2]

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

Train the model with the dataset file.

Parameters

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

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

Construct the matrix when done.

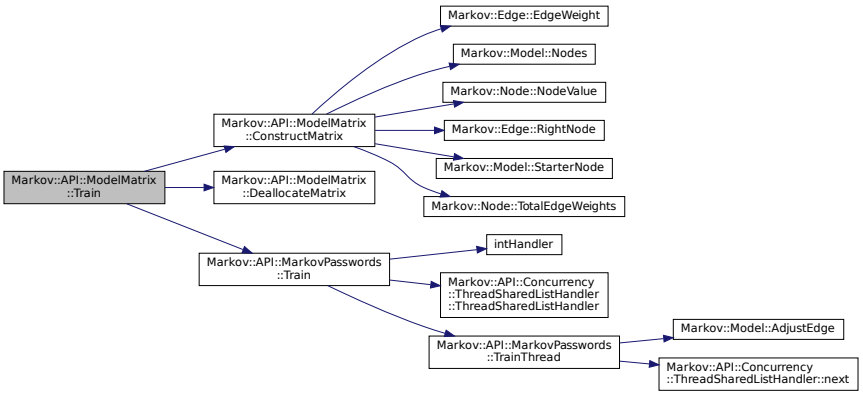
Definition at line 25 of file modelMatrix.cpp.

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

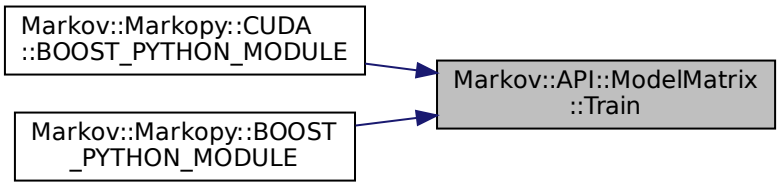
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::MarkovPasswords::Train().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.81 Train() [2/2]

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

Train the model with the dataset file.

Parameters

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

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

Construct the matrix when done.

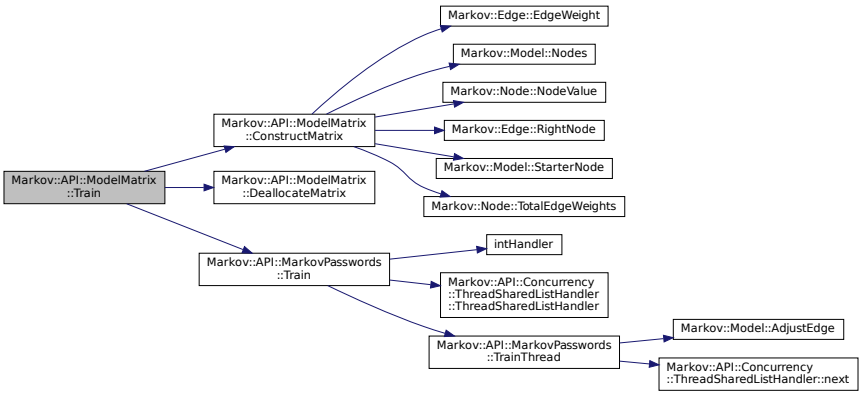
Definition at line 25 of file modelMatrix.cpp.

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

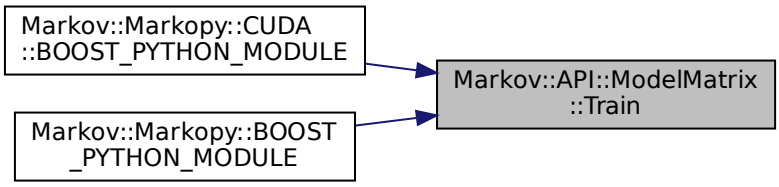
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::MarkovPasswords::Train().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.82 train() [1/2]

```
def Python.Markopy.BaseCLI.train (
               self,
              str dataset,
              str seperator,
              str output,
              bool output_forced = False,
              bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

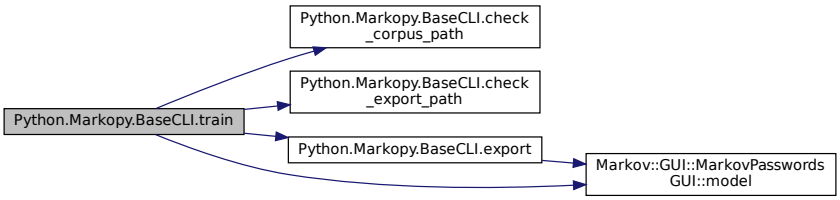
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

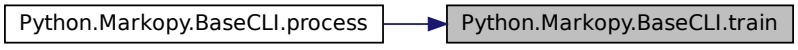
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
```

```
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.83 train() [2/2]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename

Parameters

output_forced	force overwrite
bulk	marks bulk operation with directories

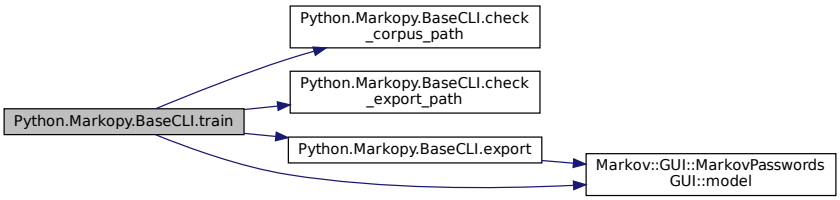
Definition at line 94 of file base.py.

```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else"} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

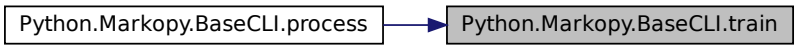
References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.12.3.84 TrainThread()

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

A single thread invoked by the Train function.

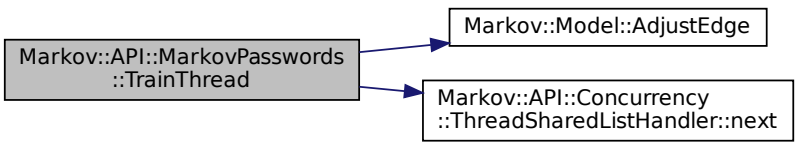
Parameters

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

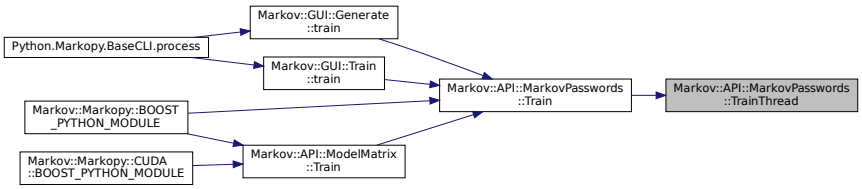
Definition at line 85 of file markovPasswords.cpp.

```
00085
                {
00086     char format_str[] ="%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
      "%ld,%s"
00097 #else
00098         sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100         this->AdjustEdge((const char*)linebuf, oc);
00101         delete linebuf;
00102     }
00103 }
```

References Markov::Model< NodeStorageType >::AdjustEdge(), keepRunning, and Markov::API::Concurrency::ThreadSharedListHa
Referenced by Markov::API::MarkovPasswords::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.12.4 Member Data Documentation

8.12.4.1 alternatingKernels

`int Markov::API::CUDA::CUDAModelMatrix::alternatingKernels [private], [inherited]`
Definition at line 135 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk().

8.12.4.2 args [1/2]

`Python.Markopy.BaseCLI.args [inherited]`
Definition at line 75 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.generate(), Python.Markopy.MarkopyCLI.help(), Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.parse(), Python.CudaMarkopy.CudaMarkopyCLI.parse_fail(), Python.Markopy.BaseCLI.process(), and Python.Markopy.BaseCLI.train().

8.12.4.3 args [2/2]

`Python.Markopy.BaseCLI.args [inherited]`
Definition at line 75 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.generate(), Python.Markopy.MarkopyCLI.help(), Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.parse(), Python.CudaMarkopy.CudaMarkopyCLI.parse_fail(), Python.Markopy.BaseCLI.process(), and Python.Markopy.BaseCLI.train().

8.12.4.4 bInfinite

`Python.CudaMarkopy.CudaModelMatrixCLI.bInfinite`
Definition at line 73 of file cudammx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate().

8.12.4.5 cudaBlocks

`int Markov::API::CUDA::CUDAModelMatrix::cudaBlocks [private], [inherited]`
Definition at line 125 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk().

8.12.4.6 cudaGridSize

`int Markov::API::CUDA::CUDAModelMatrix::cudaGridSize [private], [inherited]`
Definition at line 131 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk(), and Markov::API::CUDA::CUDAModelMatrix::prepKerne

8.12.4.7 cudaMemPerGrid

`int Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid [private], [inherited]`
Definition at line 132 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk(), and Markov::API::CUDA::CUDAModelMatrix::GatherAsy

8.12.4.8 cudaPerKernelAllocationSize

`long int Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize [private], [inherited]`
Definition at line 133 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk(), Markov::API::CUDA::CUDAModelMatrix::GatherAsyncK
and Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel().

8.12.4.9 cudastreams

`cudaStream_t* Markov::API::CUDA::CUDAModelMatrix::cudastreams [private], [inherited]`
Definition at line 139 of file cudaModelMatrix.h.

8.12.4.10 cudaThreads

`int Markov::API::CUDA::CUDAModelMatrix::cudaThreads [private], [inherited]`
Definition at line 126 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk().

8.12.4.11 datasetFile

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

8.12.4.12 device_edgeMatrix

`char* Markov::API::CUDA::CUDAModelMatrix::device_edgeMatrix [private], [inherited]`
VRAM Address pointer of edge matrix (from modelMatrix.h)
Definition at line 88 of file cudaModelMatrix.h.

8.12.4.13 device_matrixIndex

`char* Markov::API::CUDA::CUDAModelMatrix::device_matrixIndex [private], [inherited]`
VRAM Address pointer of matrixIndex (from modelMatrix.h)
Definition at line 98 of file cudaModelMatrix.h.

8.12.4.14 device_outputBuffer

`char** Markov::API::CUDA::CUDAModelMatrix::device_outputBuffer [private], [inherited]`
RandomWalk results in device.
Definition at line 108 of file cudaModelMatrix.h.

Referenced by Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel().

8.12.4.15 device_seeds

```
unsigned long** Markov::API::CUDA::CUDAModelMatrix::device_seeds  [private], [inherited]
```
Definition at line 137 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel().

8.12.4.16 device_totalEdgeWeights

```
long int* Markov::API::CUDA::CUDAModelMatrix::device_totalEdgeWeights  [private], [inherited]
```
VRAM Address pointer of total edge weights (from modelMatrix.h)
Definition at line 103 of file cudaModelMatrix.h.

8.12.4.17 device_valueMatrix

```
long int* Markov::API::CUDA::CUDAModelMatrix::device_valueMatrix  [private], [inherited]
```
VRAM Address pointer of value matrix (from modelMatrix.h)
Definition at line 93 of file cudaModelMatrix.h.

8.12.4.18 edgeMatrix [1/2]

```
char** Markov::API::ModelMatrix::edgeMatrix  [protected], [inherited]
```
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line 175 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.12.4.19 edgeMatrix [2/2]

```
char** Markov::API::ModelMatrix::edgeMatrix  [protected], [inherited]
```
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line 175 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.12.4.20 edges

```
std::vector<Edge<char >*> Markov::Model< char >::edges  [private], [inherited]
```
A list of all edges in this model.
Definition at line 204 of file model.h.

8.12.4.21 fileIO

```
Python.Markopy.ModelMatrixCLI.fileIO  [inherited]
```
Definition at line 38 of file mmx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), and Python.Markopy.ModelMatrixCLI._generate().

8.12.4.22 flatEdgeMatrix

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

Adding Edge matrix end-to-end and resize to 1-D array for better perfomance on traversing.
Definition at line 118 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix().

8.12.4.23 flatValueMatrix

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

Adding Value matrix end-to-end and resize to 1-D array for better perfomance on traversing.
Definition at line 123 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix().

8.12.4.24 iterationsPerKernelThread

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

Definition at line 127 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk().

8.12.4.25 matrixIndex [1/2]

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

to hold the Matrix index (To hold the orders of 2-D arrays')
Definition at line 190 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.12.4.26 matrixIndex [2/2]

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

to hold the Matrix index (To hold the orders of 2-D arrays')
Definition at line 190 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.12.4.27 matrixSize [1/2]

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

to hold Matrix size
Definition at line 185 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), Markov::API::ModelMatrix::FastRandomWalkThread(), and Markov::API::CUDA::CUDAMod

8.12.4.28 matrixSize [2/2]

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

to hold Matrix size
Definition at line 185 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), Markov::API::ModelMatrix::FastRandomWalkThread(), and Markov::API::CUDA::CUDAMod

8.12.4.29 model

Python.CudaMarkopy.CudaModelMatrixCLI.model
Definition at line 65 of file cudammx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.12.4.30 modelSavefile

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

Definition at line 124 of file markovPasswords.h.

8.12.4.31 nodes

std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.

8.12.4.32 numberOfPartitions

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

8.12.4.33 outputBuffer

char** Markov::API::CUDA::CUDAModelMatrix::outputBuffer [private], [inherited]
RandomWalk results in host.
Definition at line 113 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput(), and Markov::API::CUDA::CUDAModelMatrix::pr

8.12.4.34 outputFile

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

Definition at line 125 of file markovPasswords.h.

8.12.4.35 parser [1/2]

Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.12.4.36 parser [2/2]

Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.12.4.37 print_help [1/2]

Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.12.4.38 print_help [2/2]

Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.12.4.39 ready [1/2]

bool Markov::API::ModelMatrix::ready [protected], [inherited]
True when matrix is constructed. False if not.
Definition at line 200 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::ModelMatrix().

8.12.4.40 ready [2/2]

bool Markov::API::ModelMatrix::ready [protected], [inherited]
True when matrix is constructed. False if not.
Definition at line 200 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::ModelMatrix().

8.12.4.41 starterNode

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

8.12.4.42 totalEdgeWeights [1/2]

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]
Array of the Total Edge Weights.
Definition at line 195 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.12.4.43 totalEdgeWeights [2/2]

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]
Array of the Total Edge Weights.
Definition at line 195 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.12.4.44 totalOutputPerKernel

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

8.12.4.45 totalOutputPerSync

long int Markov::API::CUDA::CUDAModelMatrix::totalOutputPerSync [private], [inherited]
Definition at line 128 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk().

8.12.4.46 valueMatrix [1/2]

long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]
2-d Integer array for the value Matrix (For the weights of Edges)
Definition at line 180 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.12.4.47 valueMatrix [2/2]

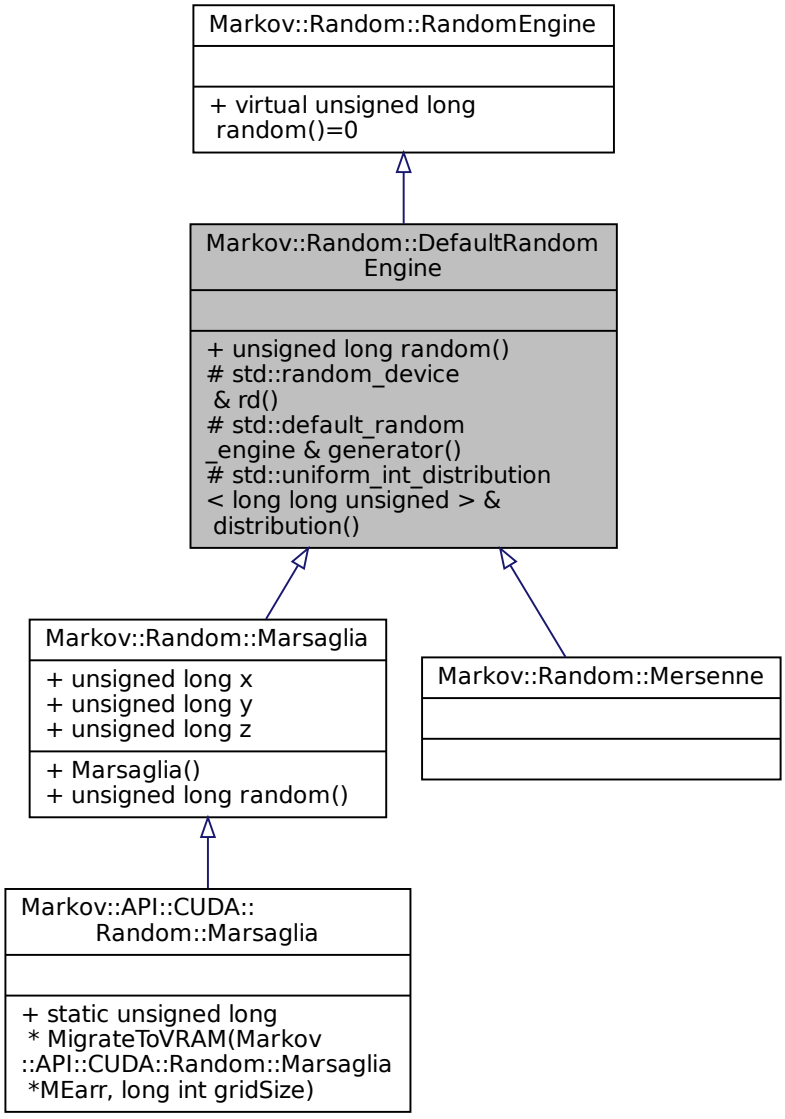
long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]
2-d Integer array for the value Matrix (For the weights of Edges)
Definition at line 180 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().
The documentation for this class was generated from the following file:

- Markopy/CudaMarkopy/src/CLI/cudammx.py

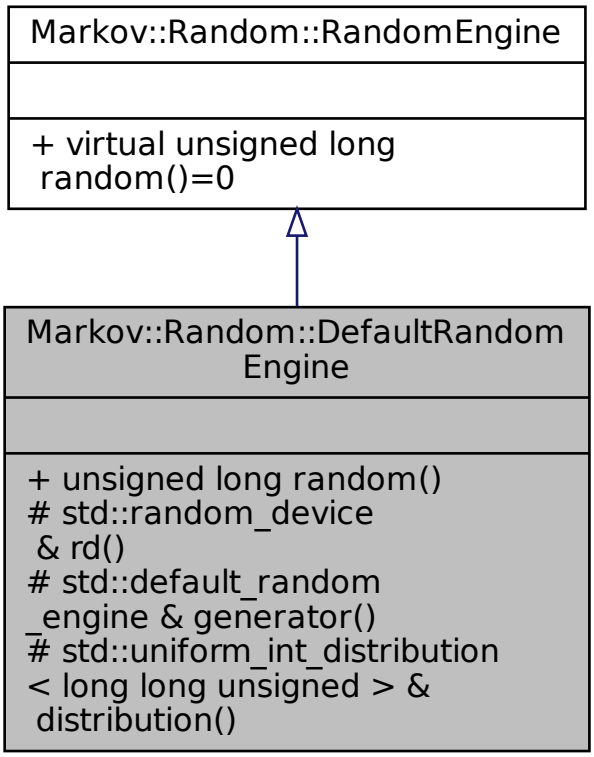
8.13 Markov::Random::DefaultRandomEngine Class Reference

Implementation using Random.h default random engine.
#include <random.h>

Inheritance diagram for Markov::Random::DefaultRandomEngine:



Collaboration diagram for Markov::Random::DefaultRandomEngine:



Public Member Functions

- unsigned long random ()

  Generate Random Number.

Protected Member Functions

- std::random_device & rd ()

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

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

  Distribution schema for seeding.

8.13.1 Detailed Description

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

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

```
    std::cout « res « "\n";
}
```

Example Use: Generating a random number with Marsaglia Engine

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

Definition at line 61 of file random.h.

8.13.2 Member Function Documentation

8.13.2.1 distribution()

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

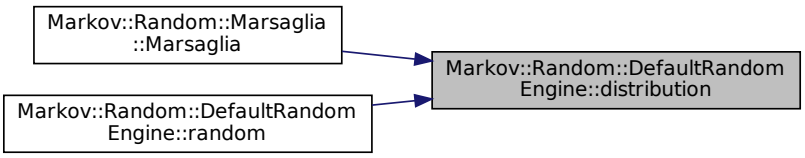
Distribution schema for seeding.

Definition at line 90 of file random.h.

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

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

Here is the caller graph for this function:



8.13.2.2 generator()

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

Default random engine for seeding.

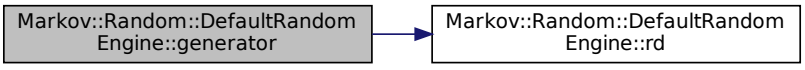
Definition at line 82 of file random.h.

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

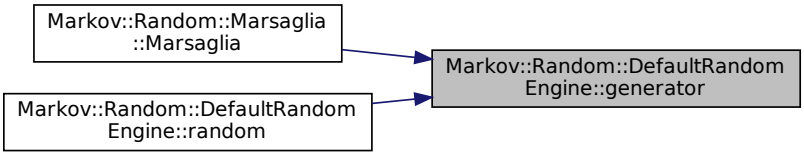
References rd().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.13.2.3 random()

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

Generate Random Number.

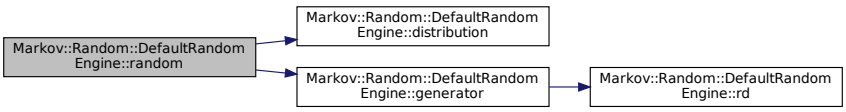
Returns

random number in long range.

Implements Markov::Random::RandomEngine.
Reimplemented in Markov::Random::Marsaglia.
Definition at line 66 of file random.h.

```
00066                                   {
00067             return this->distribution()(this->generator());
00068         }
```

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



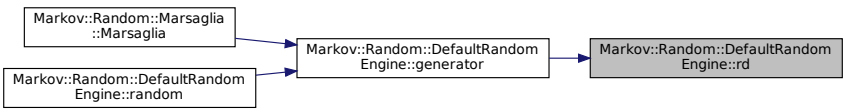
8.13.2.4 rd()

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

Default random device for seeding.
Definition at line 74 of file random.h.

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

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



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

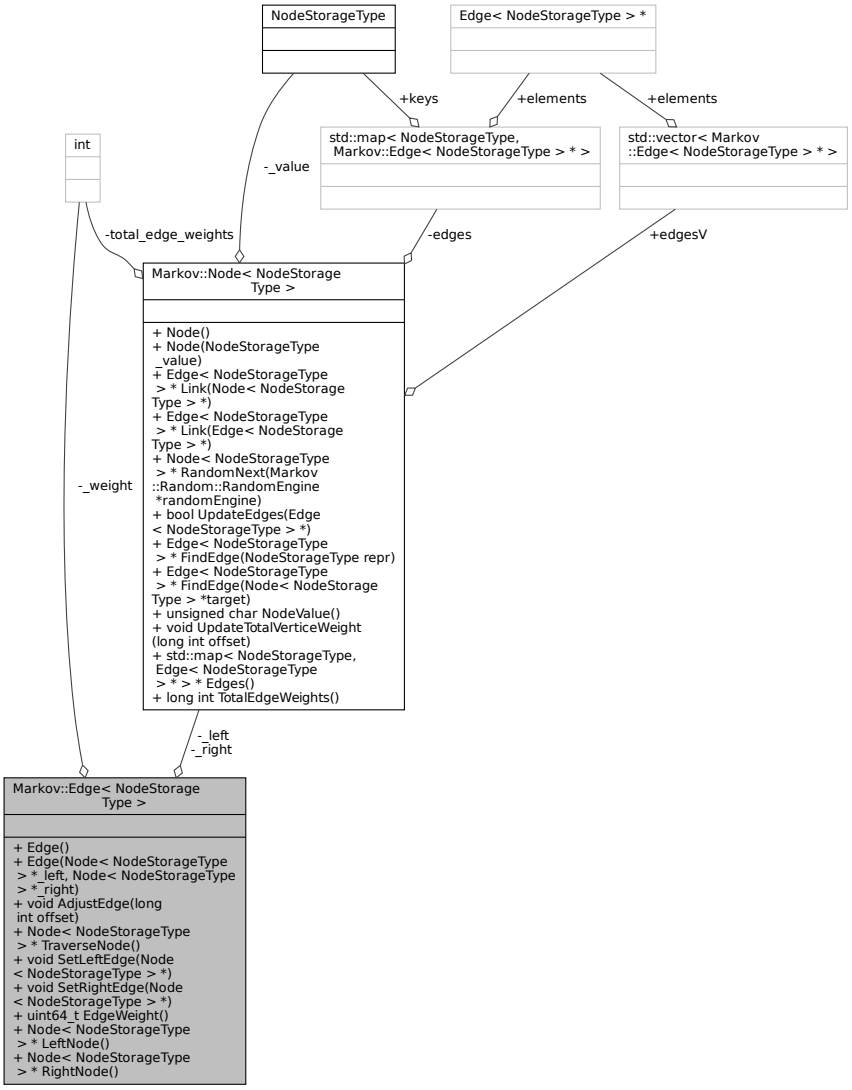
- Markopy/MarkovModel/src/random.h

8.14 Markov::Edge< NodeStorageType > Class Template Reference

Edge class used to link nodes in the model together.

`#include <edge.h>`

Collaboration diagram for Markov::Edge< NodeStorageType >:



Public Member Functions

- Edge ()

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

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

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

  Traverse this edge to RightNode.

- void SetLeftEdge (Node< NodeStorageType > *)

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

  Set RightNode of this edge.
- uint64_t EdgeWeight ()

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

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

  return edge's RightNode

Private Attributes

- Node< NodeStorageType > * _left

  source node
- Node< NodeStorageType > * _right

  target node
- long int _weight

  Edge Edge Weight.

8.14.1 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 23 of file edge.h.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 Edge() [1/2]

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

Default constructor.
Definition at line 123 of file edge.h.

```
00123                                   {
00124     this->_left = NULL;
00125     this->_right = NULL;
00126     this->_weight = 0;
00127 }
```

8.14.2.2 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 130 of file edge.h.

```
00130
        {
00131     this->_left = _left;
00132     this->_right = _right;
00133     this->_weight = 0;
00134 }
```

8.14.3 Member Function Documentation

8.14.3.1 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

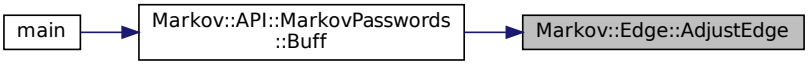
```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
e1->AdjustEdge(25);
```

Definition at line 137 of file edge.h.

```
00137                                                    {
00138     this->_weight += offset;
00139     this->LeftNode()->UpdateTotalVerticeWeight(offset);
00140 }
```

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

Here is the caller graph for this function:



8.14.3.2 EdgeWeight()

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

return edge's EdgeWeight.

Returns

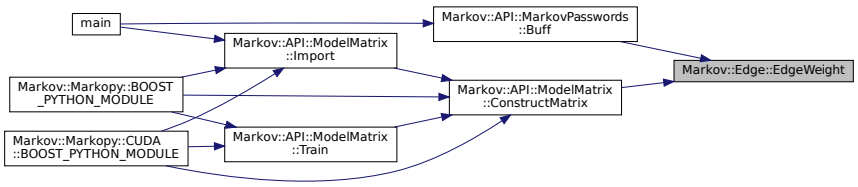
edge's EdgeWeight.

Definition at line 160 of file edge.h.

```
00160                                          {
00161     return this->_weight;
00162 }
```

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

Here is the caller graph for this function:



8.14.3.3 LeftNode()

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

return edge's LeftNode

Returns

edge's LeftNode.

Definition at line 165 of file edge.h.

```
00165                                                                {
00166     return this->_left;
00167 }
```

8.14.3.4 RightNode()

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

return edge's RightNode

Returns

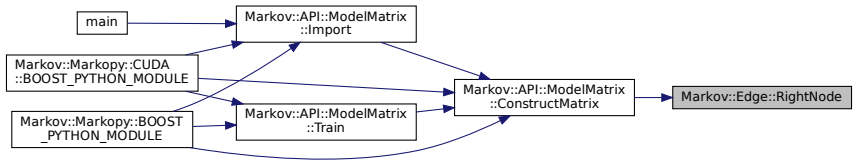
edge's RightNode.

Definition at line 170 of file edge.h.

```
00170                                                                 {
00171     return this->_right;
00172 }
```

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

Here is the caller graph for this function:



8.14.3.5 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 150 of file edge.h.

```
00150                                                                    {
00151     this->_left = n;
00152 }
```

8.14.3.6 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 155 of file edge.h.

```
00155                                                                     {
00156     this->_right = n;
00157 }
```

8.14.3.7 TraverseNode()

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

Traverse this edge to RightNode.

Returns

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

Example Use: Traverse a node

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

Definition at line 143 of file edge.h.

```
00143                                                                     {
00144     if (this->RightNode()->NodeValue() == 0xff) //terminator node
00145         return NULL;
00146     return _right;
00147 }
```

References Markov::Edge< NodeStorageType >::_right.

8.14.4 Member Data Documentation

8.14.4.1 _left

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

source node

Definition at line 105 of file edge.h.

8.14.4.2 _right

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

target node
Definition at line 110 of file edge.h.
Referenced by Markov::Edge< NodeStorageType >::TraverseNode().

8.14.4.3 _weight

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

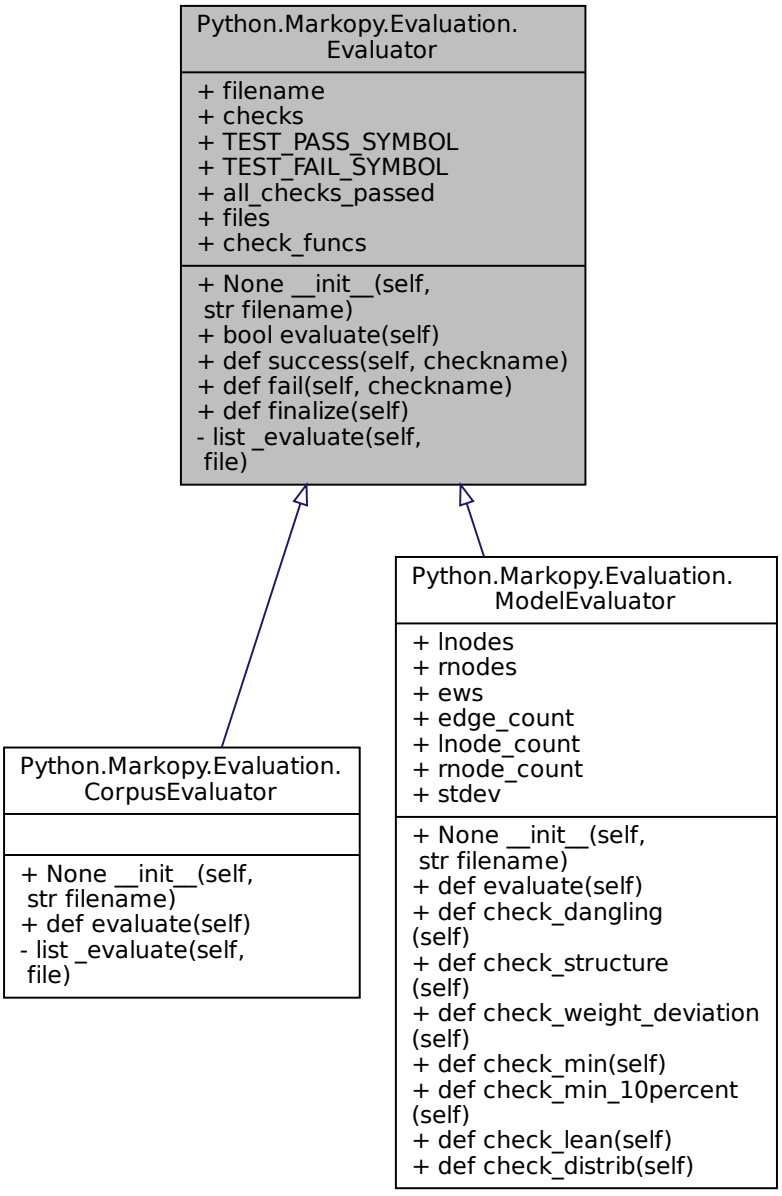
Edge Edge Weight.
Definition at line 115 of file edge.h.
The documentation for this class was generated from the following files:

- Markopy/MarkovModel/src/model.h
- Markopy/MarkovModel/src/edge.h

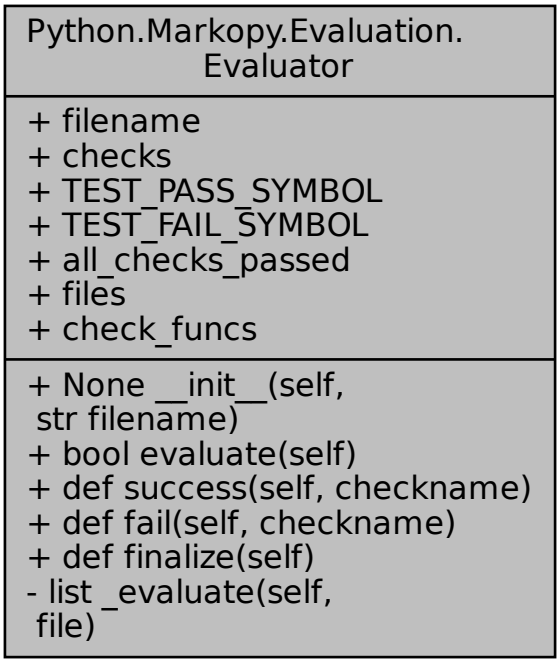
8.15 Python.Markopy.Evaluation.Evaluator Class Reference

Abstract class to evaluate and score integrity/validty.

Inheritance diagram for Python.Markopy.Evaluation.Evaluator:



Collaboration diagram for Python.Markopy.Evaluation.Evaluator:



Public Member Functions

- None __init__ (self, str filename)

  default constructor for evaluator
- bool evaluate (self)
- def success (self, checkname)

  pass a test
- def fail (self, checkname)

  fail a test
- def finalize (self)

Public Attributes

- filename
- checks
- TEST_PASS_SYMBOL
- TEST_FAIL_SYMBOL
- all_checks_passed
- files
- check_funcs

Private Member Functions

- list _evaluate (self, file)

  internal evaluation function for a single file

8.15.1 Detailed Description

Abstract class to evaluate and score integrity/validty.
Definition at line 22 of file evaluate.py.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 __init__()

```
None Python.Markopy.Evaluation.Evaluator.__init__ (
              self,
             str filename )
```

default constructor for evaluator

Parameters

filename	filename to evaluate. Can be a pattern

Reimplemented in Python.Markopy.Evaluation.CorpusEvaluator, and Python.Markopy.Evaluation.ModelEvaluator.
Definition at line 27 of file evaluate.py.

```
00027     def __init__(self, filename: str) -> None:
00028         """!
00029         @brief default constructor for evaluator
00030         @param filename filename to evaluate. Can be a pattern
00031         """
00032         self.filename = filename
00033         self.checks = []
00034         self.TEST_PASS_SYMBOL = b"\xe2\x9c\x85".decode()
00035         self.TEST_FAIL_SYMBOL = b"\xe2\x9d\x8c".decode()
00036         self.all_checks_passed = True
00037         self.files = []
00038         if("*" in filename):
00039             self.files = glob.glob(filename)
00040         else:
00041             self.files.append(filename)
00042         return True
00043
```

8.15.3 Member Function Documentation

8.15.3.1 _evaluate()

```
list Python.Markopy.Evaluation.Evaluator._evaluate (
              self,
              file )  [private]
```

internal evaluation function for a single file

Parameters

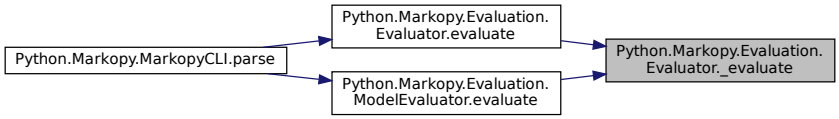
file	filename to evaluate

Reimplemented in Python.Markopy.Evaluation.CorpusEvaluator.
Definition at line 52 of file evaluate.py.

```
00052     def _evaluate(self, file) -> list:
00053         """!
00054         @brief internal evaluation function for a single file
00055         @param file filename to evaluate
00056         """
00057         if(not os.path.isfile(file)):
00058             logging.pprint(f"Given file {file} is not a valid filename")
00059             return False
00060         else:
00061              return open(file, "rb").read().split(b"\n")
00062
```

```
00063
00064
```

Referenced by Python.Markopy.Evaluation.Evaluator.evaluate(), and Python.Markopy.Evaluation.ModelEvaluator.evaluate().
Here is the caller graph for this function:



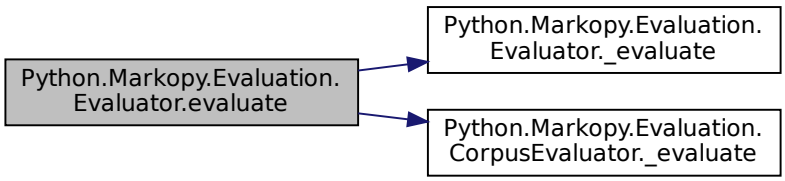
8.15.3.2 evaluate()

```
bool Python.Markopy.Evaluation.Evaluator.evaluate (
            self )
```

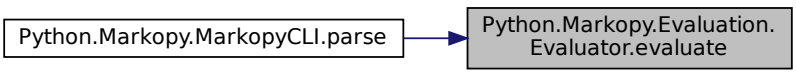
Reimplemented in Python.Markopy.Evaluation.CorpusEvaluator, and Python.Markopy.Evaluation.ModelEvaluator.
Definition at line 44 of file evaluate.py.

```
00044     def evaluate(self) -> bool:
00045         "! @brief base evaluation function"
00046         for file in self.files:
00047             self._evaluate(file)
00048
00049         self.check_funcs = [func for func in dir(self) if (callable(getattr(self, func)) and
      func.startswith("check_"))]
00050
```

References Python.Markopy.Evaluation.Evaluator._evaluate(), Python.Markopy.Evaluation.CorpusEvaluator._evaluate(), and Python.Markopy.Evaluation.Evaluator.files.
Referenced by Python.Markopy.MarkopyCLI.parse().
Here is the call graph for this function:



Here is the caller graph for this function:



8.15.3.3 fail()

```
def Python.Markopy.Evaluation.Evaluator.fail (
```

```
                self,
                checkname )
```

fail a test

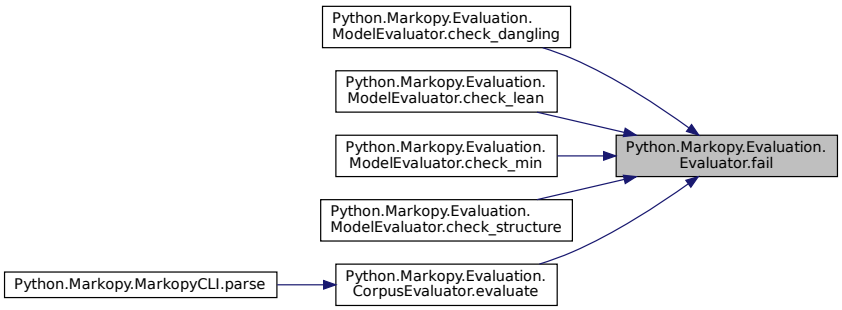
Parameters

checkname	text to display with the check

Definition at line 72 of file evaluate.py.

```
00072     def fail(self, checkname):
00073         """!
00074         @brief fail a test
00075         @param checkname text to display with the check
00076         """
00077 
00078         self.all_checks_passed = False
00079         self.checks.append((checkname, self.TEST_FAIL_SYMBOL))
00080 
```

References Python.Markopy.Evaluation.Evaluator.all_checks_passed, Python.Markopy.Evaluation.Evaluator.checks, and Python.Markopy.Evaluation.Evaluator.TEST_FAIL_SYMBOL.
Referenced by Python.Markopy.Evaluation.ModelEvaluator.check_dangling(), Python.Markopy.Evaluation.ModelEvaluator.check_lean Python.Markopy.Evaluation.ModelEvaluator.check_min(), Python.Markopy.Evaluation.ModelEvaluator.check_structure(), and Python.Markopy.Evaluation.CorpusEvaluator.evaluate().
Here is the caller graph for this function:



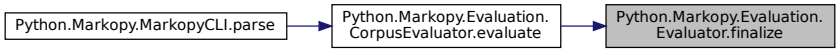
8.15.3.4 finalize()

```
def Python.Markopy.Evaluation.Evaluator.finalize (
                self )
```

Definition at line 81 of file evaluate.py.

```
00081     def finalize(self):
00082         "! @brief finalize an evaluation and print checks"
00083         print("\n################ Checks ################ ")
00084         for test in self.checks:
00085             logging.pprint(f"{test[0]:30}:{test[1]} ")
00086         print("\n")
00087         self.checks = []
00088         return self.all_checks_passed
00089 
```

References Python.Markopy.Evaluation.Evaluator.all_checks_passed, and Python.Markopy.Evaluation.Evaluator.checks.
Referenced by Python.Markopy.Evaluation.CorpusEvaluator.evaluate().
Here is the caller graph for this function:



8.15.3.5 success()

```
def Python.Markopy.Evaluation.Evaluator.success (
            self,
            checkname )
```

pass a test

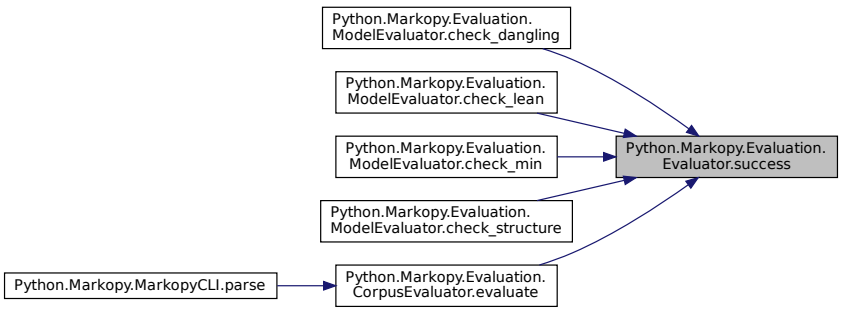
Parameters

checkname	text to display with the check

Definition at line 65 of file evaluate.py.

```
00065     def success(self, checkname):
00066         """!
00067         @brief pass a test
00068         @param checkname text to display with the check
00069         """
00070         self.checks.append((checkname, self.TEST_PASS_SYMBOL))
00071
```

References Python.Markopy.Evaluation.Evaluator.checks, and Python.Markopy.Evaluation.Evaluator.TEST_PASS_SYMBOL.
Referenced by Python.Markopy.Evaluation.ModelEvaluator.check_dangling(), Python.Markopy.Evaluation.ModelEvaluator.check_lean
Python.Markopy.Evaluation.ModelEvaluator.check_min(), Python.Markopy.Evaluation.ModelEvaluator.check_structure(),
and Python.Markopy.Evaluation.CorpusEvaluator.evaluate().
Here is the caller graph for this function:



8.15.4 Member Data Documentation

8.15.4.1 all_checks_passed

```
Python.Markopy.Evaluation.Evaluator.all_checks_passed
```

Definition at line 36 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.fail(), and Python.Markopy.Evaluation.Evaluator.finalize().

8.15.4.2 check_funcs

```
Python.Markopy.Evaluation.Evaluator.check_funcs
```

Definition at line 49 of file evaluate.py.

8.15.4.3 checks

```
Python.Markopy.Evaluation.Evaluator.checks
```

Definition at line 33 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.fail(), Python.Markopy.Evaluation.Evaluator.finalize(), and Python.Markopy.Evaluation.Evaluator.success().

8.15.4.4 filename

`Python.Markopy.Evaluation.Evaluator.filename`
Definition at line 32 of file evaluate.py.

8.15.4.5 files

`Python.Markopy.Evaluation.Evaluator.files`
Definition at line 37 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.evaluate(), Python.Markopy.Evaluation.ModelEvaluator.evaluate(), and Python.Markopy.Evaluation.CorpusEvaluator.evaluate().

8.15.4.6 TEST_FAIL_SYMBOL

`Python.Markopy.Evaluation.Evaluator.TEST_FAIL_SYMBOL`
Definition at line 35 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.fail().

8.15.4.7 TEST_PASS_SYMBOL

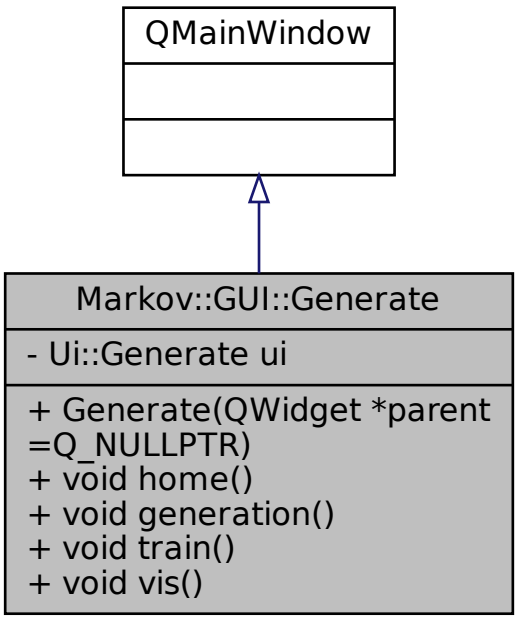
`Python.Markopy.Evaluation.Evaluator.TEST_PASS_SYMBOL`
Definition at line 34 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.success().
The documentation for this class was generated from the following file:

- Markopy/Markopy/src/CLI/evaluate.py

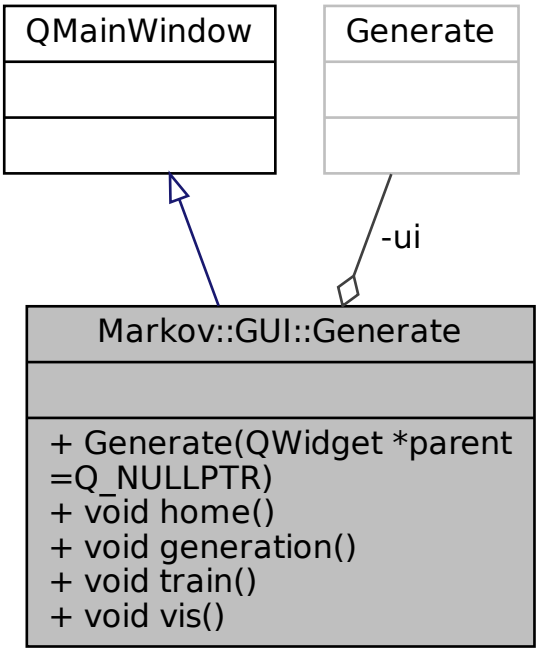
8.16 Markov::GUI::Generate Class Reference

QT Generation page class.
`#include <Generate.h>`

Inheritance diagram for Markov::GUI::Generate:



Collaboration diagram for Markov::GUI::Generate:



Public Slots

- void home ()
- void generation ()
- void train ()
- void vis ()

Public Member Functions

- Generate (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::Generate ui

8.16.1 Detailed Description

QT Generation page class.
Definition at line 15 of file Generate.h.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 Generate()

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

Definition at line 20 of file Generate.cpp.

```
00021     : QMainWindow(parent)
00022 {
00023     ui.setupUi(this);
00024
00025     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {generation(); });
00026     QObject::connect(ui.pushButton_4, &QPushButton::clicked, this, [this] {home(); });
00027     QObject::connect(ui.pushButton_3, &QPushButton::clicked, this, [this] {train(); });
00028     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {vis(); });
00029
00030
00031     ui.pushButton->setVisible(false);
00032     ui.lineEdit->setVisible(false);
00033     ui.lineEdit_2->setVisible(false);
00034     ui.lineEdit_3->setVisible(false);
00035     ui.label_3->setVisible(false);
00036     ui.label_4->setVisible(false);
00037     ui.label_5->setVisible(false);
00038
00039
00040 }
```

References ui.

8.16.3 Member Function Documentation

8.16.3.1 generation

```
void Generate::generation ( )  [slot]
```

Definition at line 42 of file Generate.cpp.

```
00042                          {
00043
00044
00045
00046
00047     QString file_name = QFileDialog::getOpenFileName(this, "Open a File", QDir::homePath());
00048     QFile file(file_name);
00049
00050
00051
00052     int numberPass = ui.lineEdit->text().toInt();
00053     int minLen = ui.lineEdit_2->text().toInt();
00054     int maxLen = ui.lineEdit_3->text().toInt();
00055     char* cstr;
00056     std::string fname = file_name.toStdString();
00057     cstr = new char[fname.size() + 1];
00058     strcpy(cstr, fname.c_str());
00059
00060     ui.label_6->setText("GENERATING!");
00061
00062     Markov::API::MarkovPasswords mp;
00063     mp.Import("src\\CLI\\sample_models\\2gram-trained.mdl");
00064
00065     mp.Generate(numberPass,cstr,minLen,maxLen);
00066
00067     if (!file.open(QFile::ReadOnly | QFile::Text)) {
00068         QMessageBox::warning(this, "Error", "File Not Open!");
00069     }
00070     QTextStream in(&file);
00071     QString text = in.readAll();
00072     ui.plainTextEdit->setPlainText(text);
00073
00074     ui.label_6->setText("DONE!");
00075
00076
00077
00078     file.close();
00079 }
```

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

Here is the call graph for this function:

8.16.3.2 home

```
void Generate::home ( )  [slot]
```

Definition at line 121 of file Generate.cpp.

```
00121                     {
00122      CLI* w = new CLI;
00123      w->show();
00124      this->close();
00125 }
```

8.16.3.3 train

```
void Generate::train ( )  [slot]
```

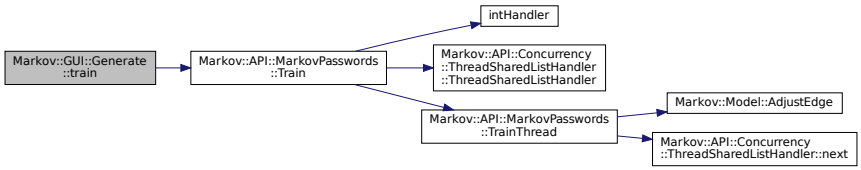
Definition at line 82 of file Generate.cpp.

```
00082                      {
00083     QString file_name = QFileDialog::getOpenFileName(this, "Open a File", QDir::homePath());
00084     QFile file(file_name);
00085
00086     if (!file.open(QFile::ReadOnly | QFile::Text)) {
00087         QMessageBox::warning(this, "Error", "File Not Open!");
00088     }
00089     QTextStream in(&file);
00090     QString text = in.readAll();
00091
00092
00093     char* cstr;
00094     std::string fname = file_name.toStdString();
00095     cstr = new char[fname.size() + 1];
00096     strcpy(cstr, fname.c_str());
00097
00098
00099
00100     char a = ',';
00101     Markov::API::MarkovPasswords mp;
00102     mp.Import("models\\2gram.mdl");
00103     mp.Train(cstr, a,10);
00104     mp.Export("models\\finished.mdl");
00105
00106
00107
00108     ui.pushButton->setVisible(true);
00109     ui.lineEdit->setVisible(true);
00110     ui.lineEdit_2->setVisible(true);
00111     ui.lineEdit_3->setVisible(true);
00112     ui.label_3->setVisible(true);
00113     ui.label_4->setVisible(true);
00114     ui.label_5->setVisible(true);
00115
00116     file.close();
00117
00118
00119 }
```

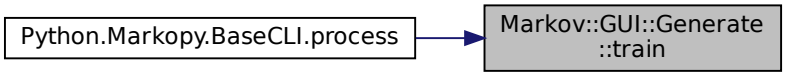
References Markov::API::MarkovPasswords::Train(), and ui.

Referenced by Python.Markopy.BaseCLI::process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.16.3.4 vis

```
void Generate::vis ( )  [slot]
```

Definition at line 126 of file Generate.cpp.

```
00126                       {
00127     MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
00128     w->show();
00129     this->close();
00130 }
```

8.16.4 Member Data Documentation

8.16.4.1 ui

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

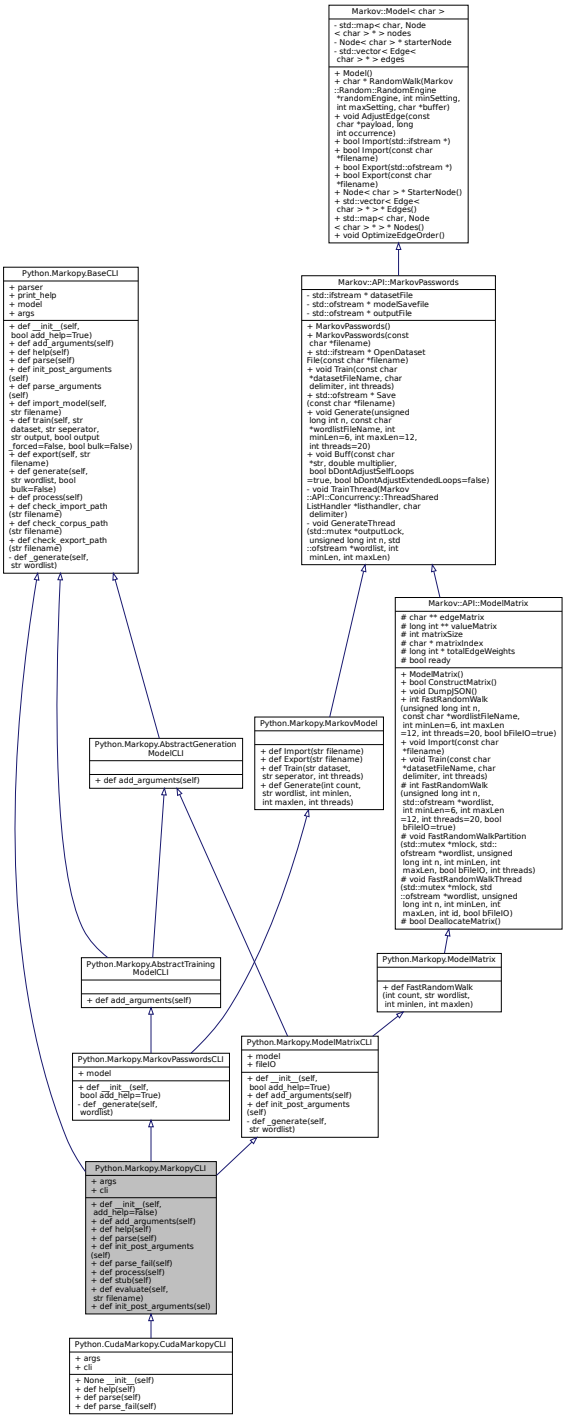
Definition at line 21 of file Generate.h.
Referenced by Generate(), generation(), and train().
The documentation for this class was generated from the following files:

- Markopy/MarkovPasswordsGUI/src/Generate.h
- Markopy/MarkovPasswordsGUI/src/Generate.cpp

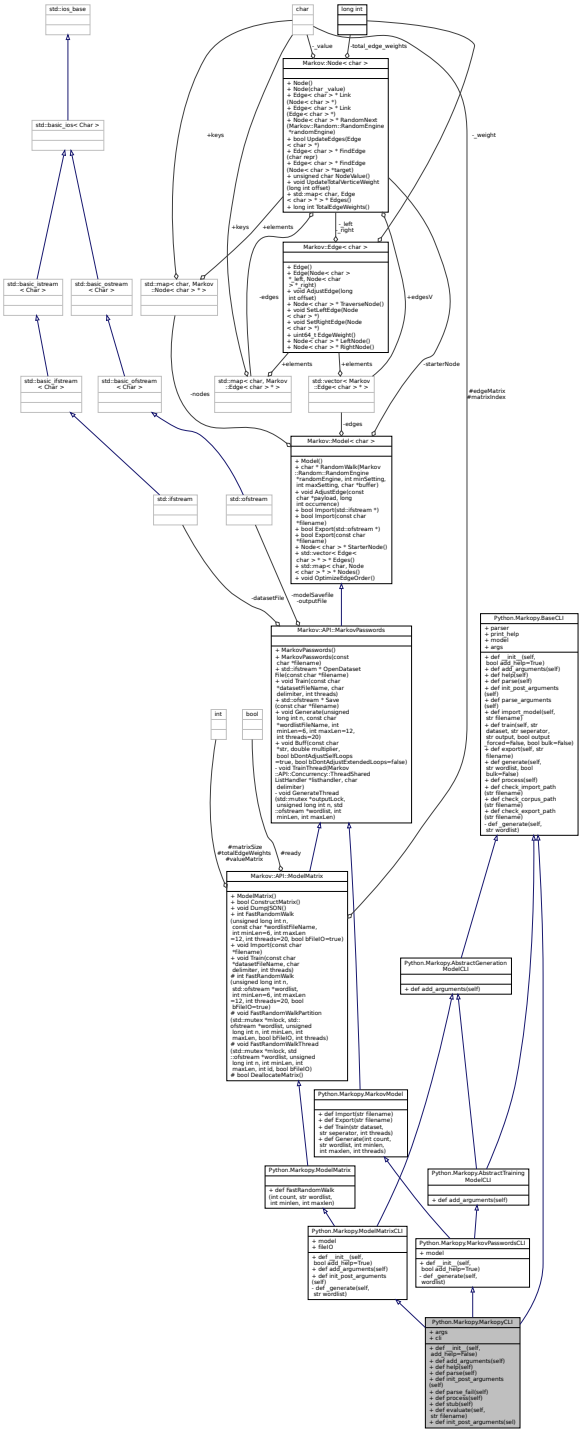
8.17 Python.Markopy.MarkopyCLI Class Reference

Top level model selector for Markopy CLI.

Inheritance diagram for Python.Markopy.MarkopyCLI:



Collaboration diagram for Python.Markopy.MarkopyCLI:



Public Member Functions

- def __init__ (self, add_help=False)

  default constructor

- def add_arguments (self)

  add -mt/–model_type constructor

- def help (self)

  overload help function to print submodel helps

- def parse (self)
- def init_post_arguments (self)
- def parse_fail (self)
- def process (self)

  Process parameters for operation.
- def stub (self)
- def evaluate (self, str filename)
- def init_post_arguments (sel)
- def parse_arguments (self)
- def import_model (self, str filename)

  Import a model file.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def parse_arguments (self)
- def import_model (self, str filename)

  Import a model file.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def FastRandomWalk (int count, str wordlist, int minlen, int maxlen)
- int FastRandomWalk (unsigned long int n, const char ∗wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

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

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

  Debug function to dump the model to a JSON file.
- void Import (const char ∗filename)

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

  Import a file to construct the model.
- void Train (const char ∗datasetFileName, char delimiter, int threads)

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

  Open dataset file and return the ifstream pointer.
- std::ofstream ∗ Save (const char ∗filename)

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

  Call Markov::Model::RandomWalk n times, and collect output.
- void Buff (const char ∗str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- char ∗ RandomWalk (Markov::Random::RandomEngine ∗randomEngine, int minSetting, int maxSetting, char ∗buffer)

Do a random walk on this model.

- void AdjustEdge (const char *payload, long int occurrence)

  Adjust the model with a single string.
- bool Export (std::ofstream *)

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

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

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

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

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.
- def parse_arguments (self)
- def parse_arguments (self)
- def import_model (self, str filename)

  Import a model file.
- def import_model (self, str filename)

  Import a model file.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def Import (str filename)
- bool Import (std::ifstream *)

  Import a file to construct the model.
- def Export (str filename)
- 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.
- def Train (str dataset, str seperator, int threads)
- def Generate (int count, str wordlist, int minlen, int maxlen, int threads)
- 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.
- std::ifstream * OpenDatasetFile (const char *filename)

  Open dataset file and return the ifstream pointer.
- std::ofstream * Save (const char *filename)

  Export model to file.

- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- char * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

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

  Adjust the model with a single string.
- Node< char > * StarterNode ()

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

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

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.

Static Public Member Functions

- def check_import_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- def check_import_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- def check_import_path (str filename)

  check import path for validity
- def check_import_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity

Public Attributes

- args
- cli
- parser
- print_help
- model
- model

- fileIO
- parser
- print_help
- model
- parser
- parser
- print_help
- print_help

Protected Member Functions

- int FastRandomWalk (unsigned long int n, std::ofstream ∗wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- void FastRandomWalkPartition (std::mutex ∗mlock, std::ofstream ∗wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

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

  A single thread of a single partition of FastRandomWalk.
- bool DeallocateMatrix ()

  Deallocate matrix and make it ready for re-construction.

Protected Attributes

- char ∗∗ edgeMatrix

  2-D Character array for the edge Matrix (The characters of Nodes)
- long int ∗∗ valueMatrix

  2-d Integer array for the value Matrix (For the weights of Edges)
- int matrixSize

  to hold Matrix size
- char ∗ matrixIndex

  to hold the Matrix index (To hold the orders of 2-D arrays')
- long int ∗ totalEdgeWeights

  Array of the Total Edge Weights.
- bool ready

  True when matrix is constructed. False if not.

Private Member Functions

- def _generate (self, str wordlist)

  wrapper for generate function.
- 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

  Dataset file input of our system

- std::ofstream ∗ outputFile

  File to save model of our system

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

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

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

  A list of all edges in this model.

8.17.1 Detailed Description

Top level model selector for Markopy CLI.
This class is used for injecting the -mt parameter to the CLI, and determining the model type depending on that.
Definition at line 56 of file markopy.py.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 __init__()

```
def Python.Markopy.MarkopyCLI.__init__ (
              self,
              add_help = False )
```

default constructor
Definition at line 66 of file markopy.py.

```
00066     def __init__(self, add_help=False):
00067         """!
00068         @brief default constructor
00069         """
00070 
00071         BaseCLI.__init__(self,add_help)
00072         self.args = None
00073         self.parser.epilog = f"""
00074         {colored("Sample runs:", "yellow")}
00075         {__file__.split("/")[-1]} -mt MP generate trained.mdl -n 500 -w output.txt
00076             Import trained.mdl, and generate 500 lines to output.txt
00077 
00078         {__file__.split("/")[-1]} -mt MMX generate trained.mdl -n 500 -w output.txt
00079             Import trained.mdl, and generate 500 lines to output.txt
00080         """
00081 
```

8.17.3 Member Function Documentation

8.17.3.1 _generate()

```
def Python.Markopy.BaseCLI._generate (
              self,
             str wordlist )  [private], [inherited]
```

wrapper for generate function.
This can be overloaded by other models

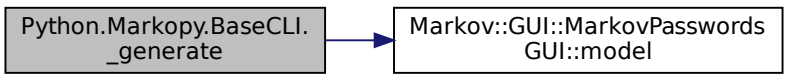
Parameters

wordlist	filename to generate to

Reimplemented in Python.Markopy.ModelMatrixCLI, and Python.CudaMarkopy.CudaModelMatrixCLI.
Definition at line 161 of file base.py.

```
00161     def _generate(self, wordlist : str):
00162         """!
00163         @brief wrapper for generate function. This can be overloaded by other models
00164         @param wordlist filename to generate to
00165         """
00166         self.model.Generate(int(self.args.count), wordlist, int(self.args.min), int(self.args.max),
      int(self.args.threads))
00167
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.generate().
Here is the call graph for this function:



Here is the caller graph for this function:

8.17.3.2 add_arguments()

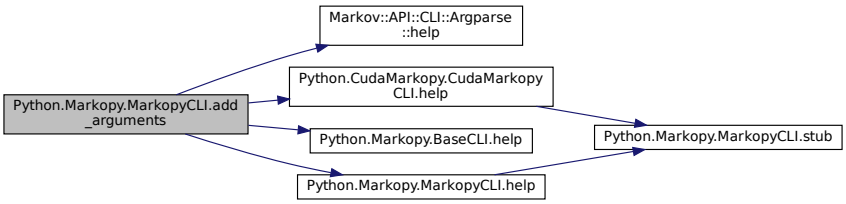
```
def Python.Markopy.MarkopyCLI.add_arguments (
              self )
```

add -mt/–model_type constructor
Reimplemented from Python.Markopy.BaseCLI.
Definition at line 83 of file markopy.py.

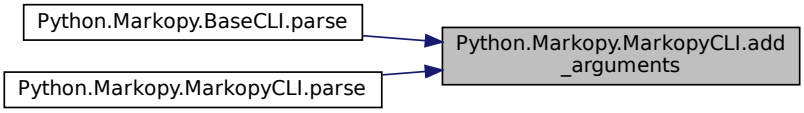
```
00083     def add_arguments(self):
00084         """!
00085         @brief add -mt/--model_type constructor
00086         """
00087         self.parser.add_argument("-mt", "--model_type", default="_MMX", help="Model type to use.
      Accepted values: MP, MMX")
00088         self.parser.add_argument("-h", "--help", action="store_true", help="Model type to use.
      Accepted values: MP, MMX")
00089         self.parser.add_argument("-ev", "--evaluate", help="Evaluate a models integrity")
00090         self.parser.add_argument("-evt", "--evaluate_type", help="Evaluation type, model or corpus")
00091         self.parser.print_help = self.help
00092
```

References Markov::API::CLI::Argparse.help(), Python.CudaMarkopy.CudaMarkopyCLI.help(), Python.Markopy.BaseCLI.help(), Python.Markopy.MarkopyCLI.help(), and Python.Markopy.BaseCLI.parser.
Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.3 AdjustEdge() [1/2]

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

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

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

Parameters

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

Definition at line 109 of file model.h.

```
00337                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
```

```
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.17.3.4 AdjustEdge() [2/2]

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

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

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

Parameters

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

Definition at line 109 of file model.h.

```
00337                                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.17.3.5 Buff() [1/2]

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

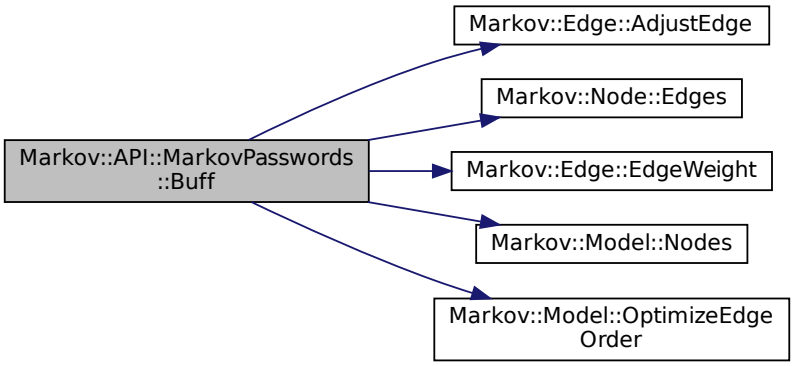
Definition at line 153 of file markovPasswords.cpp.

```
00153
                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173
00174         }
00175         i++;
00176     }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

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

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.17.3.6 Buff() [2/2]

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

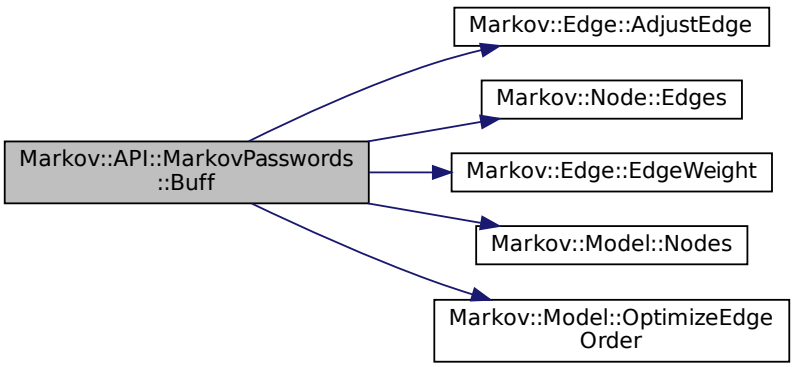
Definition at line 153 of file markovPasswords.cpp.

```
00153                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173
00174         }
00175         i++;
00176     }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

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

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.17.3.7 check_corpus_path() [1/4]

```
def Python.Markopy.BaseCLI.check_corpus_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186 
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.17.3.8 check_corpus_path() [2/4]

```
def Python.Markopy.BaseCLI.check_corpus_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
```

```
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.17.3.9 check_corpus_path() [3/4]

```
def Python.Markopy.BaseCLI.check_corpus_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.17.3.10 check_corpus_path() [4/4]

```
def Python.Markopy.BaseCLI.check_corpus_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
```

```
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.17.3.11 check_export_path() [1/4]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.17.3.12 check_export_path() [2/4]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
```

```
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.17.3.13 check_export_path() [3/4]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.17.3.14 check_export_path() [4/4]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
```

```
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.17.3.15 check_import_path() [1/4]

```
def Python.Markopy.BaseCLI.check_import_path (
             str filename )  [static], [inherited]
```

check import path for validity

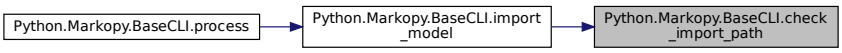
Parameters

filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:



8.17.3.16 check_import_path() [2/4]

```
def Python.Markopy.BaseCLI.check_import_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

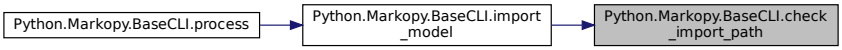
filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
```

```
00172         @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:



8.17.3.17 check_import_path() [3/4]

```
def Python.Markopy.BaseCLI.check_import_path (
            str filename )  [static], [inherited]
```

check import path for validity

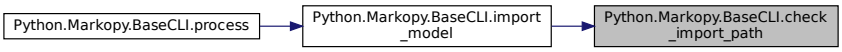
Parameters

filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:



8.17.3.18 check_import_path() [4/4]

```
def Python.Markopy.BaseCLI.check_import_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
```

```
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:

8.17.3.19 ConstructMatrix()

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

Construct the related Matrix data for the model.
This operation can be used after importing/training to allocate and populate the matrix content.
this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

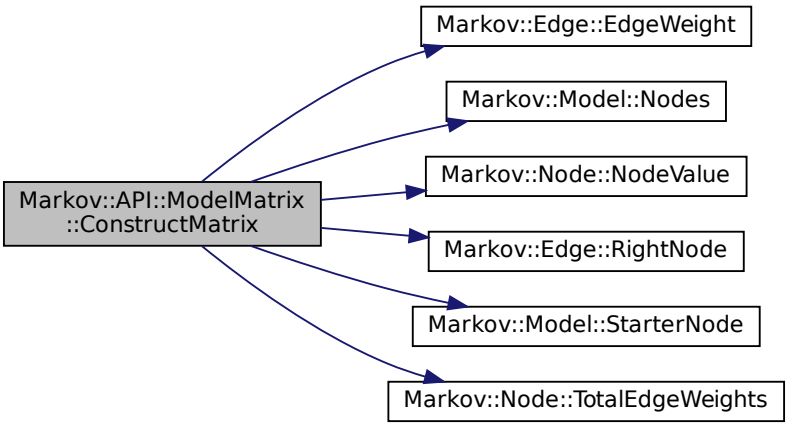
True if constructed. False if already construced.

Definition at line 31 of file modelMatrix.cpp.

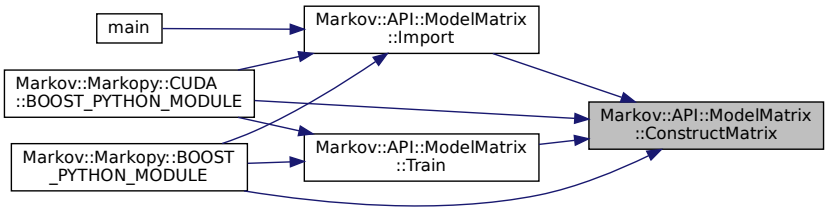
```
00031                                           {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034 
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037 
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
```

```
00071             }
00072
00073         }
00074         i++;
00075     }
00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::API::ModelMatrix::r
Markov::API::ModelMatrix::matrixSize, Markov::Model< NodeStorageType >::Nodes(), Markov::Node< storageType >::NodeValue()
Markov::API::ModelMatrix::ready, Markov::Edge< NodeStorageType >::RightNode(), Markov::Model< NodeStorageType >::StarterN
Markov::API::ModelMatrix::totalEdgeWeights, Markov::Node< storageType >::TotalEdgeWeights(), and Markov::API::ModelMatrix::va
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(),
Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.20 DeallocateMatrix()

```
bool Markov::API::ModelMatrix::DeallocateMatrix ( )  [protected], [inherited]
```

Deallocate matrix and make it ready for re-construction.

Returns

True if deallocated. False if matrix was not initialized

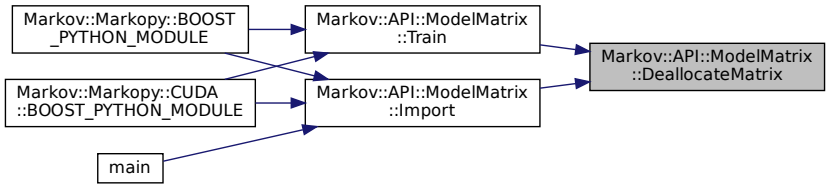
Definition at line 81 of file modelMatrix.cpp.

```
00081                                                {
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
00085 
00086     for(int i=0;i<this->matrixSize;i++){
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090 
00091     for(int i=0;i<this->matrixSize;i++){
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095 
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
00099 }
```

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

Referenced by Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().

Here is the caller graph for this function:



8.17.3.21 DumpJSON()

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

Debug function to dump the model to a JSON file.

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

Definition at line 101 of file modelMatrix.cpp.

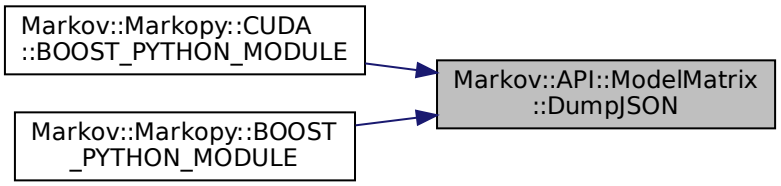
```
00101                                     {
00102 
00103     std::cout « "{\n    \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout « "\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout « "\\\\";
00107         else if(this->matrixIndex[i]==0) std::cout « "\\\\x00";
00108         else if(i==0) std::cout « "\\\\xff";
00109         else if(this->matrixIndex[i]=='\n') std::cout « "\\n";
00110         else std::cout « this->matrixIndex[i];
00111     }
00112     std::cout «
00113     "\",\n"
00114     "    \"edgemap\": {\n";
00115 
00116     for(int i=0;i<this->matrixSize;i++){
00117         if(this->matrixIndex[i]=='"') std::cout « "        \"\\\"\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout « "        \"\\\\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout « "        \"\\\\x00\": [";
00120         else if(this->matrixIndex[i]<0) std::cout « "        \"\\\\xff\": [";
00121         else std::cout « "        \"" « this->matrixIndex[i] « "\": [";
00122         for(int j=0;j<this->matrixSize;j++){
00123             if(this->edgeMatrix[i][j]=='"') std::cout « "\"\\\"\"";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout « "\"\\\\\"";
00125             else if(this->edgeMatrix[i][j]==0) std::cout « "\"\\\\x00\"";
00126             else if(this->edgeMatrix[i][j]<0) std::cout « "\"\\\\xff\"";
00127             else if(this->matrixIndex[i]=='\n') std::cout « "\"\\n\"";
```

```
00128             else std::cout « "\"" « this->edgeMatrix[i][j] « "\"";
00129             if(j!=this->matrixSize-1) std::cout « ", ";
00130         }
00131         std::cout « "],\n";
00132     }
00133     std::cout « "},\n";
00134
00135     std::cout « "\"   weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++){
00137         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00140         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00141         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++){
00144             std::cout « this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout « ", ";
00146         }
00147         std::cout « "],\n";
00148     }
00149     std::cout « "  }\n}\n";
00150 }
```

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

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

Here is the caller graph for this function:



8.17.3.22 Edges() [1/2]

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

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.17.3.23 Edges() [2/2]

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

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.17.3.24 evaluate()

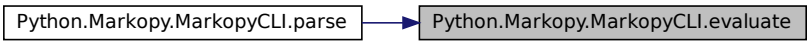
```
def Python.Markopy.MarkopyCLI.evaluate (
              self,
             str filename )
```

Definition at line 163 of file markopy.py.

```
00163     def evaluate(self,filename : str):
00164         if(not self.args.evaluate_type):
00165             if(filename.endswith(".mdl")):
00166                 ModelEvaluator(filename).evaluate()
00167             else:
00168                 CorpusEvaluator(filename).evaluate()
00169         else:
00170             if(self.args.evaluate_type == "model"):
00171                 ModelEvaluator(filename).evaluate()
00172             else:
00173                 CorpusEvaluator(filename).evaluate()
00174
```

Referenced by Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.17.3.25 Export() [1/5]

```
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 166 of file model.h.

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

8.17.3.26 Export() [2/5]

```
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 166 of file model.h.

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

8.17.3.27 export() [1/4]

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

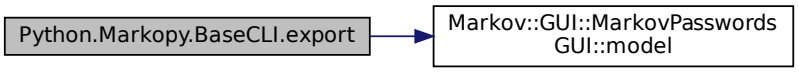
Parameters

filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.17.3.28 export() [2/4]

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

Parameters

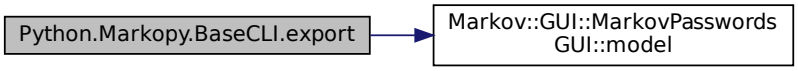
filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.17.3.29 export() [3/4]

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

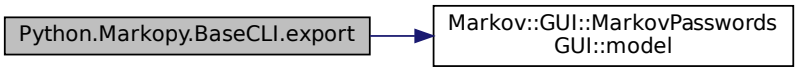
Parameters

filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.17.3.30 export() [4/4]

```
def Python.Markopy.BaseCLI.export (
             self,
            str filename )  [inherited]
```

Export model to a file.

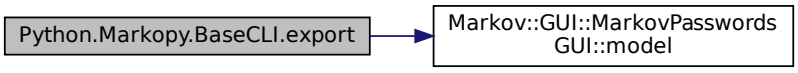
Parameters

filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.17.3.31 Export() [3/5]

```
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 155 of file model.h.

```
00288                                                    {
```

```
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
      e->RightNode()->NodeValue() « "\n";
00293         *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.17.3.32 Export() [4/5]

```
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 155 of file model.h.

```
00288                                                 {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
      e->RightNode()->NodeValue() « "\n";
00293         *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.17.3.33 Export() [5/5]

```
def Python.Markopy.MarkovModel.Export (
            str filename )  [inherited]
```

Definition at line 26 of file mm.py.

```
00026     def Export(filename : str):
00027         pass
00028
```

8.17.3.34 FastRandomWalk() [1/3]

```
def Python.Markopy.ModelMatrix.FastRandomWalk (
            int count,
            str wordlist,
            int minlen,
            int maxlen )  [inherited]
```

Definition at line 48 of file mm.py.

```
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass
```

Referenced by Python.Markopy.ModelMatrixCLI._generate().

Here is the caller graph for this function:

8.17.3.35 FastRandomWalk() [2/3]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [inherited]
```

Random walk on the Matrix-reduced Markov::Model.
This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.
While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

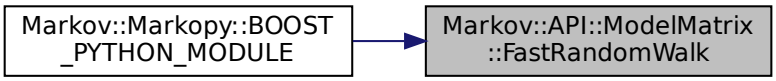
```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 217 of file modelMatrix.cpp.

```
00217
                                            {
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

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

Here is the caller graph for this function:



8.17.3.36 FastRandomWalk() [3/3]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [protected], [inherited]
```

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

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5

This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

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

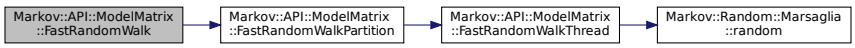
Definition at line 204 of file modelMatrix.cpp.

```
00204
                                    {
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
      threads);
00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
      threads);
00213     }
00214     return 0;
00215 }
```

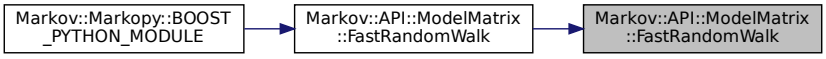
References Markov::API::ModelMatrix::FastRandomWalkPartition().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.37 FastRandomWalkPartition()

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

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

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

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

Parameters

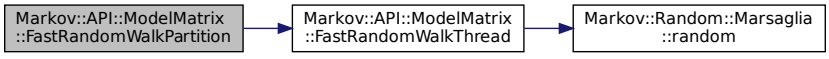
mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads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

Definition at line 225 of file modelMatrix.cpp.

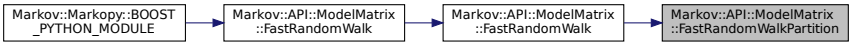
```
00225
                                                                {
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
```

```
00232     int id = 0;
00233     for(int i=0;i<threads;i++){
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
      mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
      wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00239
00240     for(int i=0;i<threads;i++){
00241         threadsV[i]->join();
00242     }
00243 }
```

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



Here is the caller graph for this function:



8.17.3.38 FastRandomWalkThread()

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

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

Parameters

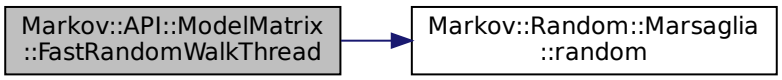
mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
id	- **DEPRECATED** Thread id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

Definition at line 153 of file modelMatrix.cpp.

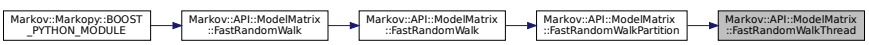
```
00153
```

```
                                                    {
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179
00180             if (len >= maxLen)  break;
00181             else if ((next < 0) && (len < minLen)) continue;
00182             else if (next < 0) break;
00183             cur = next;
00184             res[bufferctr + len++] = cur;
00185         }
00186         res[bufferctr + len++] = '\n';
00187         bufferctr+=len;
00188
00189     }
00190     if(bFileIO){
00191         mlock->lock();
00192         *wordlist << res;
00193         mlock->unlock();
00194     }else{
00195         mlock->lock();
00196         std::cout << res;
00197         mlock->unlock();
00198     }
00199     delete res;
00200
00201 }
```

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



Here is the caller graph for this function:



8.17.3.39 Generate() [1/3]

```
def Python.Markopy.MarkovModel.Generate (
```

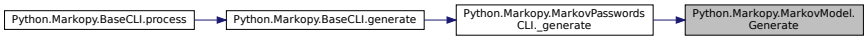
```
            int count,
            str wordlist,
            int minlen,
            int maxlen,
            int threads )  [inherited]
```

Definition at line 34 of file mm.py.

```
00034     def Generate(count : int, wordlist : str, minlen : int, maxlen: int, threads : int):
00035         pass
00036
00037
```

Referenced by Python.Markopy.MarkovPasswordsCLI._generate().

Here is the caller graph for this function:



8.17.3.40 generate() [1/4]

```
def Python.Markopy.BaseCLI.generate (
             self,
            str wordlist,
            bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

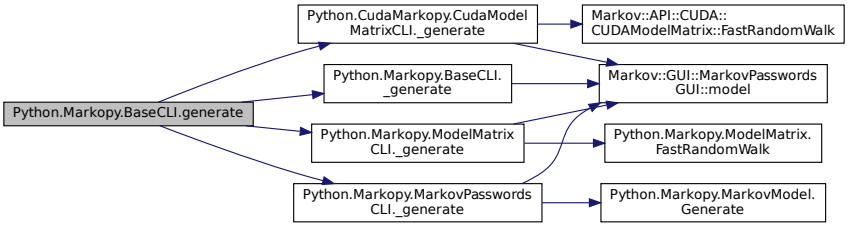
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.17.3.41 generate() [2/4]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

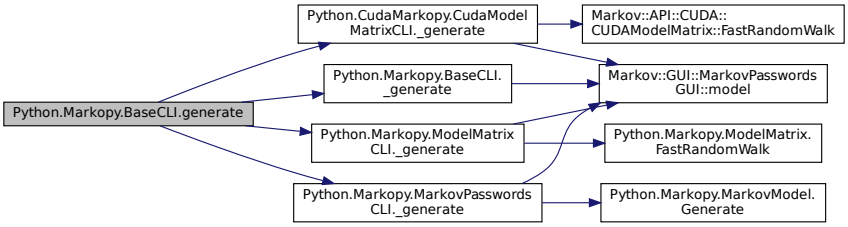
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159 
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.17.3.42 generate() [3/4]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

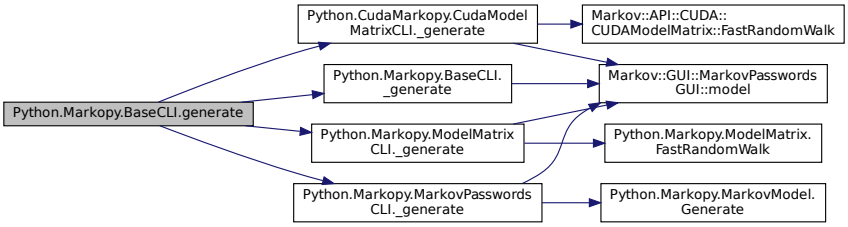
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159 
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.17.3.43 generate() [4/4]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

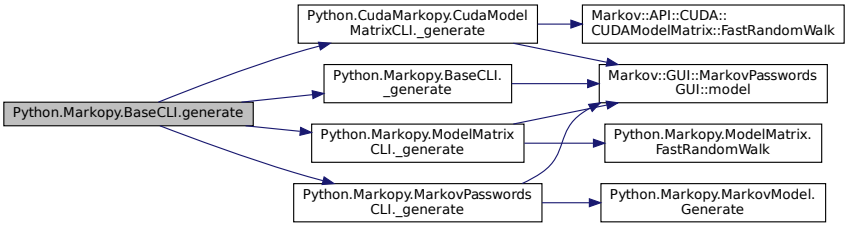
Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159 
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.17.3.44 Generate() [2/3]

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

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

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

Parameters

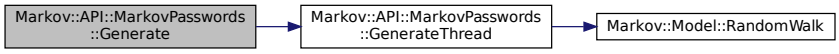
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

Definition at line 118 of file markovPasswords.cpp.

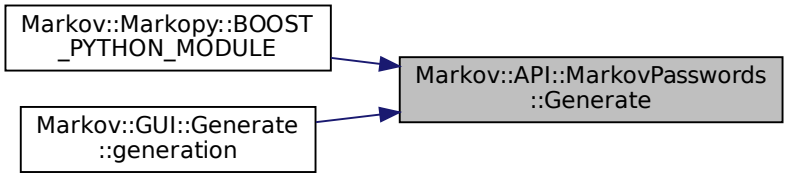
```
00118                                {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
       &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
```

```
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

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



Here is the caller graph for this function:



8.17.3.45 Generate() [3/3]

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

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

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

Parameters

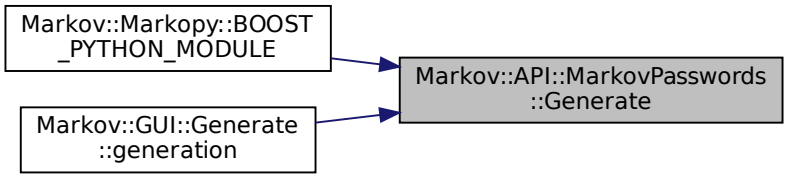
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

Definition at line 118 of file markovPasswords.cpp.

```
00118
                                {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
      &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

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

Here is the caller graph for this function:



8.17.3.46 GenerateThread()

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

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

Parameters

outputLock	- shared mutex lock to lock during output operation. Prevents race condition on write.

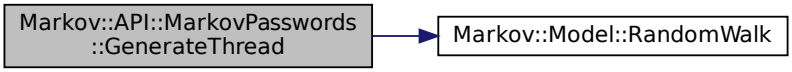
Parameters

n	number of lines to be generated by this thread
wordlist	wordlistfile
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

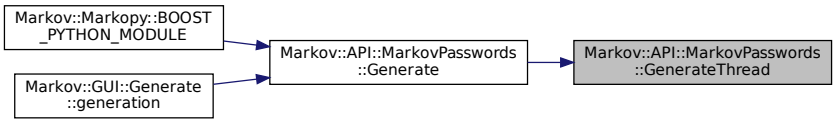
Definition at line 140 of file markovPasswords.cpp.

```
00140                                       {
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143 
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist « res « "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

References Markov::Model< NodeStorageType >::RandomWalk().
Referenced by Markov::API::MarkovPasswords::Generate().
Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.47 help()

```
def Python.Markopy.MarkopyCLI.help (
              self )
```

overload help function to print submodel helps
Reimplemented from Python.Markopy.BaseCLI.
Reimplemented in Python.CudaMarkopy.CudaMarkopyCLI.
Definition at line 94 of file markopy.py.

```
00094     def help(self):
00095         """!
00096         @brief overload help function to print submodel helps
00097         """
00098         self.parser.print_help = self.stub
00099         self.args = self.parser.parse_known_args()[0]
00100         if(self.args.model_type!="_MMX"):
00101             if(self.args.model_type=="MP"):
00102                 mp = MarkovPasswordsCLI()
00103                 mp.add_arguments()
00104                 mp.parser.print_help()
```

```
00105             elif(self.args.model_type=="MMX"):
00106                 mp = ModelMatrixCLI()
00107                 mp.add_arguments()
00108                 mp.parser.print_help()
00109         else:
00110             print(colored("Model Mode selection choices:", "green"))
00111             self.print_help()
00112             print(colored("Following are applicable for -mt MP mode:", "green"))
00113             mp = MarkovPasswordsCLI()
00114             mp.add_arguments()
00115             mp.parser.print_help()
00116             print(colored("Following are applicable for -mt MMX mode:", "green"))
00117             mp = ModelMatrixCLI()
00118             mp.add_arguments()
00119             mp.parser.print_help()
00120
00121         exit()
00122
00123
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.parser, Python.Markopy.BaseCLI.print_help, and Python.Markopy.MarkopyCLI.stub().
Referenced by Python.Markopy.MarkopyCLI.add_arguments().
Here is the call graph for this function:

Here is the caller graph for this function:

8.17.3.48 Import() [1/4]

```
void Markov::API::ModelMatrix::Import (
            const char * filename )  [inherited]
```

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

Returns

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

Example Use: Import a file with filename

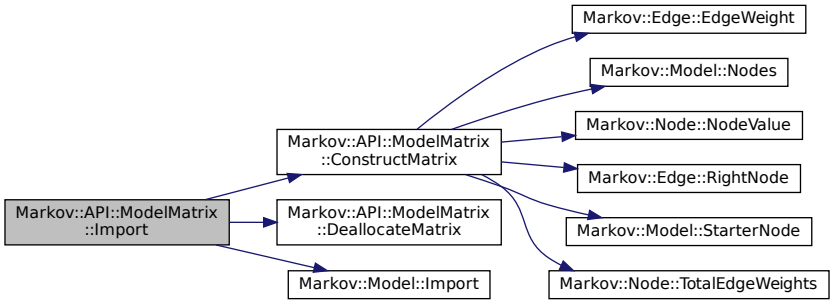
```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.
Definition at line 19 of file modelMatrix.cpp.

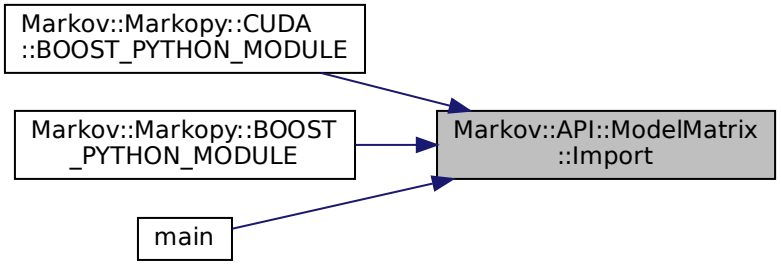
```
00019                                                 {
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::Model< NodeStorageType >::Import().
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.49 Import() [2/4]

```
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 126 of file model.h.

```
00216                                                    {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout « "cell: " « cell « std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
```

```
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout << "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
      int(targetN->NodeValue()) << "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

8.17.3.50 Import() [3/4]

```
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 126 of file model.h.

```
00216                                                   {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
```

```
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout « "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

8.17.3.51 Import() [4/4]

```
def Python.Markopy.MarkovModel.Import (
             str filename )  [inherited]
```

Definition at line 22 of file mm.py.

```
00022     def Import(filename : str):
00023         pass
00024
```

8.17.3.52 import_model() [1/4]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

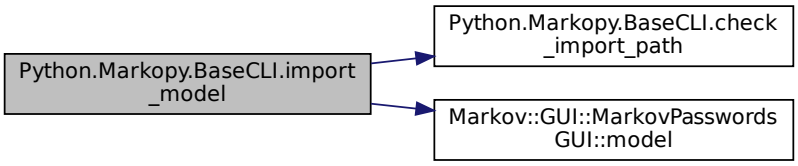
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

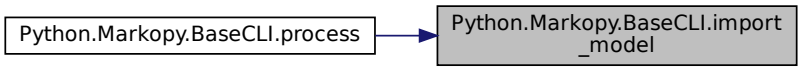
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.53 import_model() [2/4]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

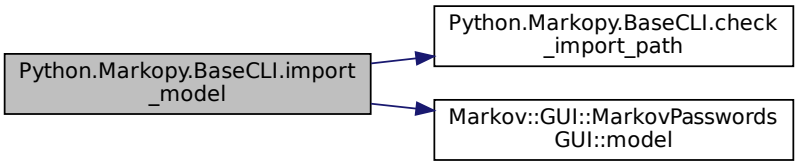
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

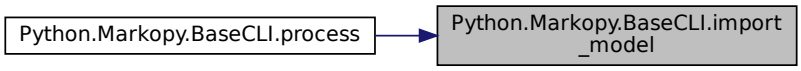
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.54 import_model() [3/4]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

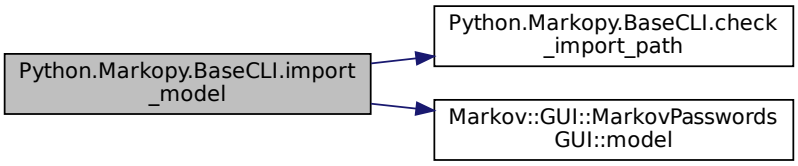
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

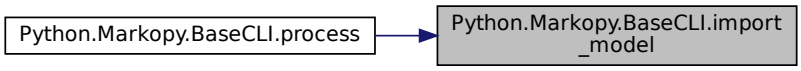
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.55 import_model() [4/4]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

filename	filename to import

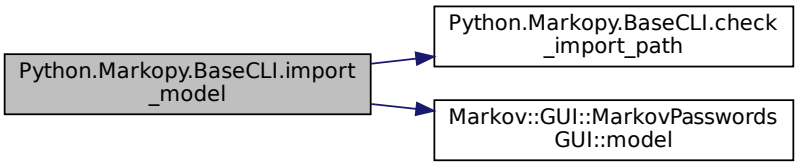
Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083 
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087 
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091 
00092 
00093 
```

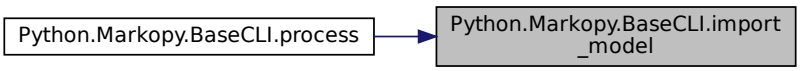
References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.56 init_post_arguments() [1/2]

```
def Python.Markopy.MarkopyCLI.init_post_arguments (
            sel )
```

Reimplemented from Python.Markopy.BaseCLI.

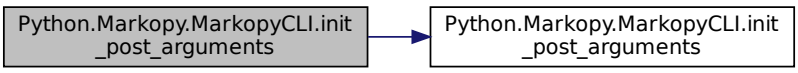
Definition at line 175 of file markopy.py.

```
00175     def init_post_arguments(sel):
00176         pass
00177
```

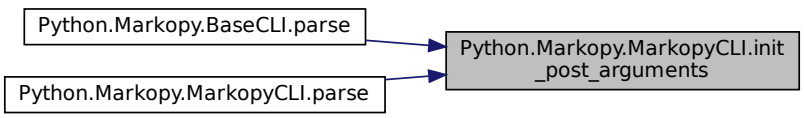
References Python.Markopy.MarkopyCLI.init_post_arguments().

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.57 init_post_arguments() [2/2]

```
def Python.Markopy.MarkopyCLI.init_post_arguments (
            self )
```

Reimplemented from Python.Markopy.BaseCLI.

Definition at line 145 of file markopy.py.

```
00145     def init_post_arguments(self):
00146         pass
00147
```

Referenced by Python.Markopy.MarkopyCLI.init_post_arguments(), Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:

8.17.3.58 Nodes() [1/2]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.17.3.59 Nodes() [2/2]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.17.3.60 OpenDatasetFile() [1/2]

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

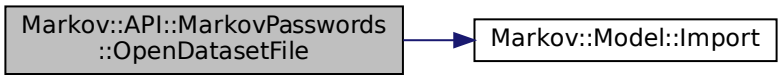
Returns

ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References Markov::Model< NodeStorageType >::Import().
Here is the call graph for this function:



8.17.3.61 OpenDatasetFile() [2/2]

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

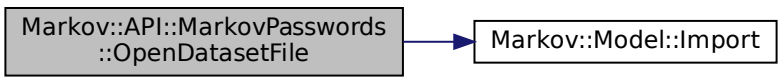
Returns

ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References Markov::Model< NodeStorageType >::Import().
Here is the call graph for this function:



8.17.3.62 OptimizeEdgeOrder() [1/2]

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

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.17.3.63 OptimizeEdgeOrder() [2/2]

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

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.17.3.64 parse()

```
def Python.Markopy.MarkopyCLI.parse (
            self )
```

Reimplemented from Python.Markopy.BaseCLI.

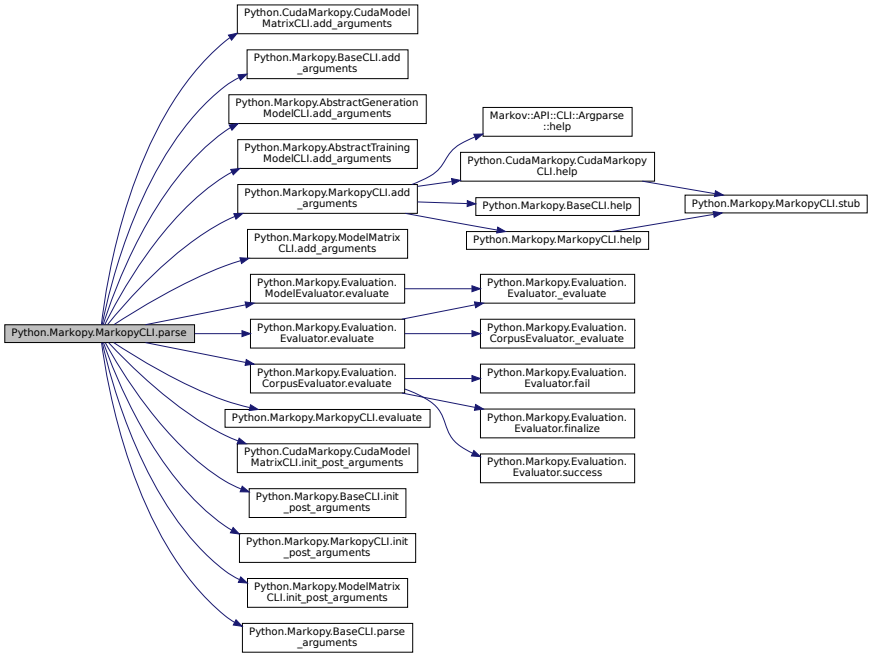
Reimplemented in Python.CudaMarkopy.CudaMarkopyCLI.

Definition at line 125 of file markopy.py.

```
00125     def parse(self):
00126         "! @brief overload parse function to parse for submodels"
00127 
00128         self.add_arguments()
00129         self.parse_arguments()
00130         self.init_post_arguments()
00131         if(self.args.evaluate):
00132             self.evaluate(self.args.evaluate)
00133             exit()
00134         if(self.args.model_type == "MP"):
00135             self.cli = MarkovPasswordsCLI()
00136         elif(self.args.model_type == "MMX" or self.args.model_type == "_MMX"):
00137             self.cli = ModelMatrixCLI()
00138         else:
00139             self.parse_fail()
00140 
00141         if(self.args.help): return self.help()
00142         self.cli.parse()
00143 
```

References Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarko Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.Evaluation.Evaluator.evaluate(), Python.Markopy.Evaluation.ModelEvaluator.evaluate(), Python.Markopy.Evaluation.CorpusEvaluator.evaluate(), Python.Markopy.MarkopyCLI.evaluate(), Python.CudaMarkopy.CudaModelMatrixCLI.init_post_arguments(),

Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.init_post_arguments(), Python.Markopy.ModelMatrixC
and Python.Markopy.BaseCLI.parse_arguments().
Here is the call graph for this function:



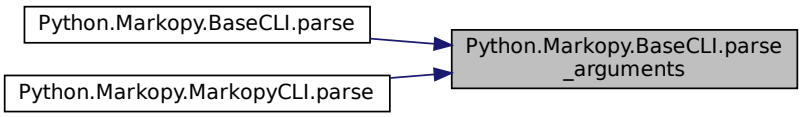
8.17.3.65 parse_arguments() [1/4]

```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().
Here is the caller graph for this function:



8.17.3.66 parse_arguments() [2/4]

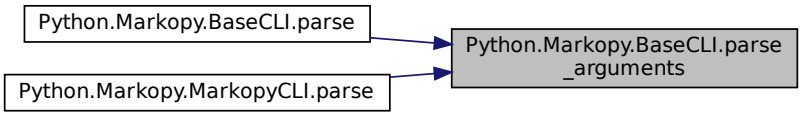
```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
```

```
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().
Here is the caller graph for this function:



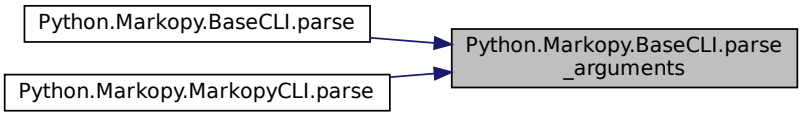
8.17.3.67 parse_arguments() [3/4]

```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().
Here is the caller graph for this function:



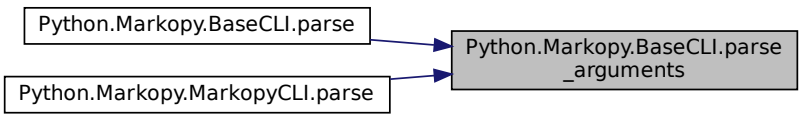
8.17.3.68 parse_arguments() [4/4]

```
def Python.Markopy.BaseCLI.parse_arguments (
              self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().
Here is the caller graph for this function:



8.17.3.69 parse_fail()

```
def Python.Markopy.MarkopyCLI.parse_fail (
              self )
```

Reimplemented in Python.CudaMarkopy.CudaMarkopyCLI.

Definition at line 149 of file markopy.py.

```
00149     def parse_fail(self):
00150         "! @brief failed to parse model type"
00151         print("Unrecognized model type.")
00152         exit()
00153
```

8.17.3.70 process()

```
def Python.Markopy.MarkopyCLI.process (
              self )
```

Process parameters for operation.

Reimplemented from Python.Markopy.BaseCLI.

Definition at line 154 of file markopy.py.

```
00154     def process(self):
00155         "! @brief pass the process request to selected submodel"
00156         return self.cli.process()
00157
```

References Python.CudaMarkopy.CudaMarkopyCLI.cli, and Python.Markopy.MarkopyCLI.cli.

8.17.3.71 RandomWalk() [1/2]

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307                                                           {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319 
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323 
00324         n = temp_node;
00325 
00326         buffer[len++] = n->NodeValue();
00327     }
00328 
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331 
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.17.3.72 RandomWalk() [2/2]

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307                                                       {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319 
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323 
00324         n = temp_node;
00325 
00326         buffer[len++] = n->NodeValue();
00327     }
00328 
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331 
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.17.3.73 Save() [1/2]

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

Export model to file.

Parameters

filename	- Export filename.

Returns

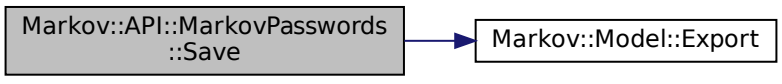
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                   {
00107     std::ofstream* exportFile;
00108 
00109     std::ofstream newFile(filename);
00110 
00111     exportFile = &newFile;
00112 
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

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

Here is the call graph for this function:



8.17.3.74 Save() [2/2]

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

Export model to file.

Parameters

filename	- Export filename.

Returns

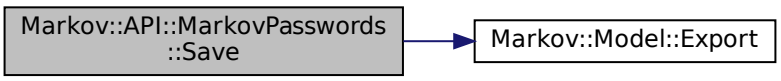
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                    {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

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

Here is the call graph for this function:



8.17.3.75 StarterNode() [1/2]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.17.3.76 StarterNode() [2/2]

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.17.3.77 stub()

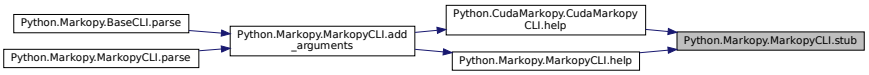
```
def Python.Markopy.MarkopyCLI.stub (
              self )
```

Definition at line 158 of file markopy.py.

```
00158     def stub(self):
00159         "! @brief stub function to hack help requests"
00160         return
00161
00162
```

Referenced by Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.help().

Here is the caller graph for this function:



8.17.3.78 Train() [1/2]

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

Train the model with the dataset file.

Parameters

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

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

Construct the matrix when done.

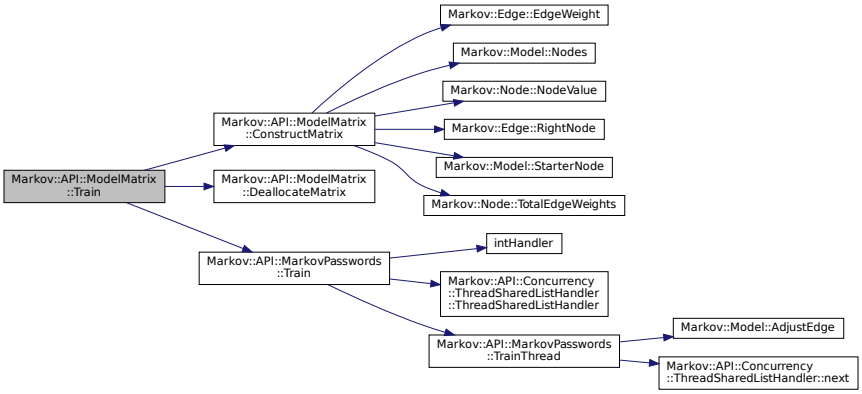
Definition at line 25 of file modelMatrix.cpp.

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

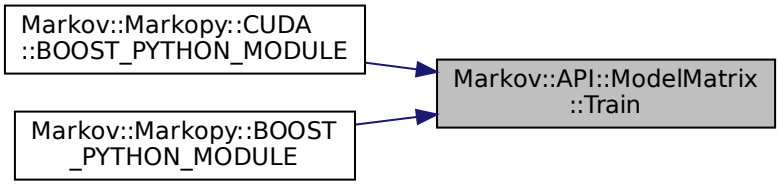
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::MarkovPasswords::Train().

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.79 train() [1/4]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

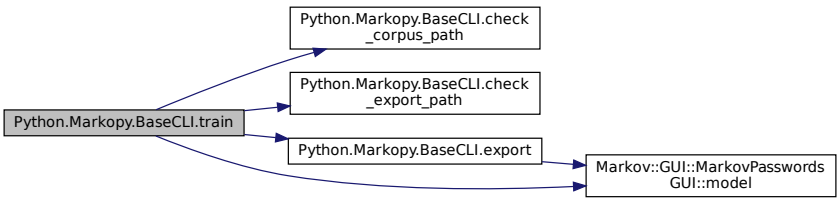
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
       bool=False):
```

```
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

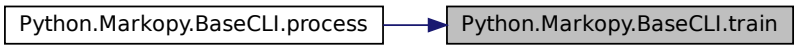
References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.80 train() [2/4]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

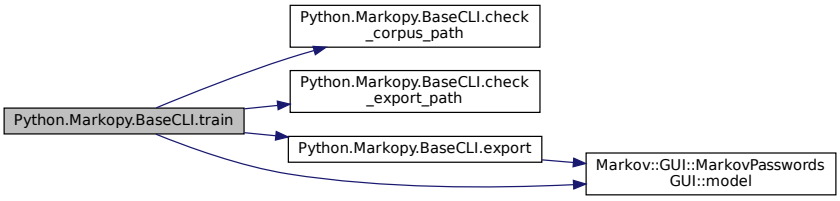
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

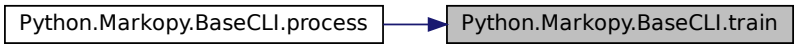
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
       bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
       else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
       accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.81 train() [3/4]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

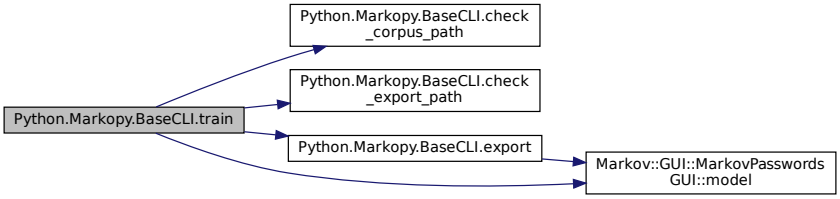
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

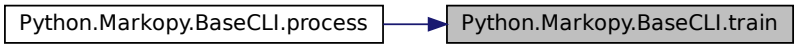
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
```

```
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.82 train() [4/4]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

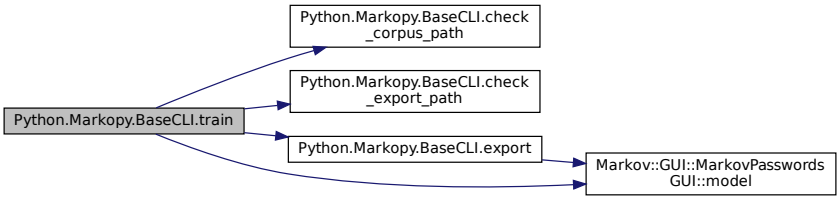
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

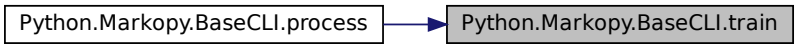
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.17.3.83 Train() [2/2]

```
def Python.Markopy.MarkovModel.Train (
            str dataset,
            str seperator,
            int threads )  [inherited]
```

Definition at line 30 of file mm.py.

```
00030     def Train(dataset: str, seperator : str, threads : int):
00031         pass
00032
```

8.17.3.84 TrainThread()

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

A single thread invoked by the Train function.

Parameters

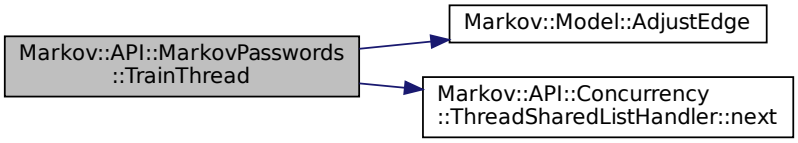
listhandler	- Listhandler class to read corpus from
delimiter	- a character, same as the delimiter in dataset content

Definition at line 85 of file markovPasswords.cpp.

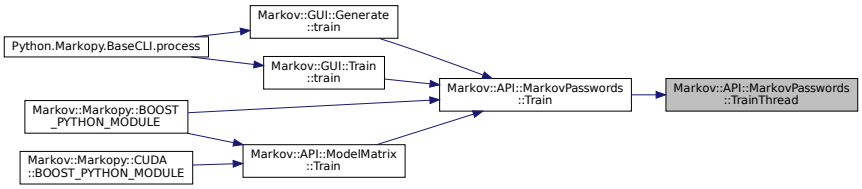
```
00085
                {
00086     char format_str[] ="%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
      "%ld,%s"
```

```
00097 #else
00098         sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100         this->AdjustEdge((const char*)linebuf, oc);
00101         delete linebuf;
00102     }
00103 }
```

References Markov::Model< NodeStorageType >::AdjustEdge(), keepRunning, and Markov::API::Concurrency::ThreadSharedListHa
Referenced by Markov::API::MarkovPasswords::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.17.4 Member Data Documentation

8.17.4.1 args

`Python.Markopy.MarkopyCLI.args`
Definition at line 72 of file markopy.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.generate Python.Markopy.MarkopyCLI.help(), Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.parse(), Python.CudaMarkopy.CudaMarkopyCLI.parse_fail(), Python.Markopy.BaseCLI.process(), and Python.Markopy.BaseCLI.train().

8.17.4.2 cli

`Python.Markopy.MarkopyCLI.cli`
Definition at line 135 of file markopy.py.
Referenced by Python.Markopy.MarkopyCLI.process().

8.17.4.3 datasetFile

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

8.17.4.4 edgeMatrix

`char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]`
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line 175 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.17.4.5 edges

`std::vector<Edge<char >*> Markov::Model< char >::edges [private], [inherited]`
A list of all edges in this model.
Definition at line 204 of file model.h.

8.17.4.6 fileIO

`Python.Markopy.ModelMatrixCLI.fileIO [inherited]`
Definition at line 38 of file mmx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), and Python.Markopy.ModelMatrixCLI._generate().

8.17.4.7 matrixIndex

`char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]`
to hold the Matrix index (To hold the orders of 2-D arrays')
Definition at line 190 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.17.4.8 matrixSize

`int Markov::API::ModelMatrix::matrixSize [protected], [inherited]`
to hold Matrix size
Definition at line 185 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), Markov::API::ModelMatrix::FastRandomWalkThread(), and Markov::API::CUDA::CUDAMod

8.17.4.9 model [1/3]

`Python.Markopy.BaseCLI.model [inherited]`
Definition at line 40 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.17.4.10 model [2/3]

`Python.Markopy.ModelMatrixCLI.model [inherited]`
Definition at line 30 of file mmx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.17.4.11 model [3/3]

Python.Markopy.MarkovPasswordsCLI.model [inherited]
Definition at line 29 of file mp.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.17.4.12 modelSavefile

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

Definition at line 124 of file markovPasswords.h.

8.17.4.13 nodes

std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.

8.17.4.14 outputFile

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

Definition at line 125 of file markovPasswords.h.

8.17.4.15 parser [1/4]

Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.17.4.16 parser [2/4]

Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.17.4.17 parser [3/4]

Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.17.4.18 parser [4/4]

`Python.Markopy.BaseCLI.parser [inherited]`
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.17.4.19 print_help [1/4]

`Python.Markopy.BaseCLI.print_help [inherited]`
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.17.4.20 print_help [2/4]

`Python.Markopy.BaseCLI.print_help [inherited]`
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.17.4.21 print_help [3/4]

`Python.Markopy.BaseCLI.print_help [inherited]`
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.17.4.22 print_help [4/4]

`Python.Markopy.BaseCLI.print_help [inherited]`
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.17.4.23 ready

`bool Markov::API::ModelMatrix::ready [protected], [inherited]`
True when matrix is constructed. False if not.
Definition at line 200 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::ModelMatrix().

8.17.4.24 starterNode

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

8.17.4.25 totalEdgeWeights

`long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]`
Array of the Total Edge Weights.
Definition at line 195 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.17.4.26 valueMatrix

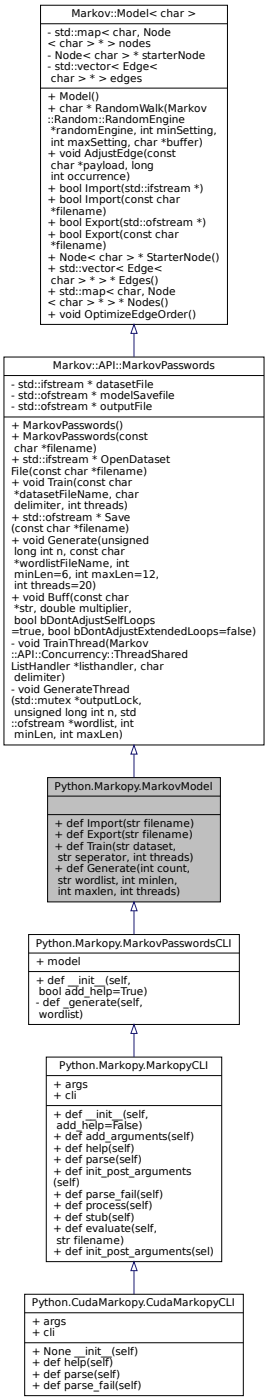
`long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]`
2-d Integer array for the value Matrix (For the weights of Edges)
Definition at line 180 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().
The documentation for this class was generated from the following file:

- Markopy/Markopy/src/CLI/markopy.py

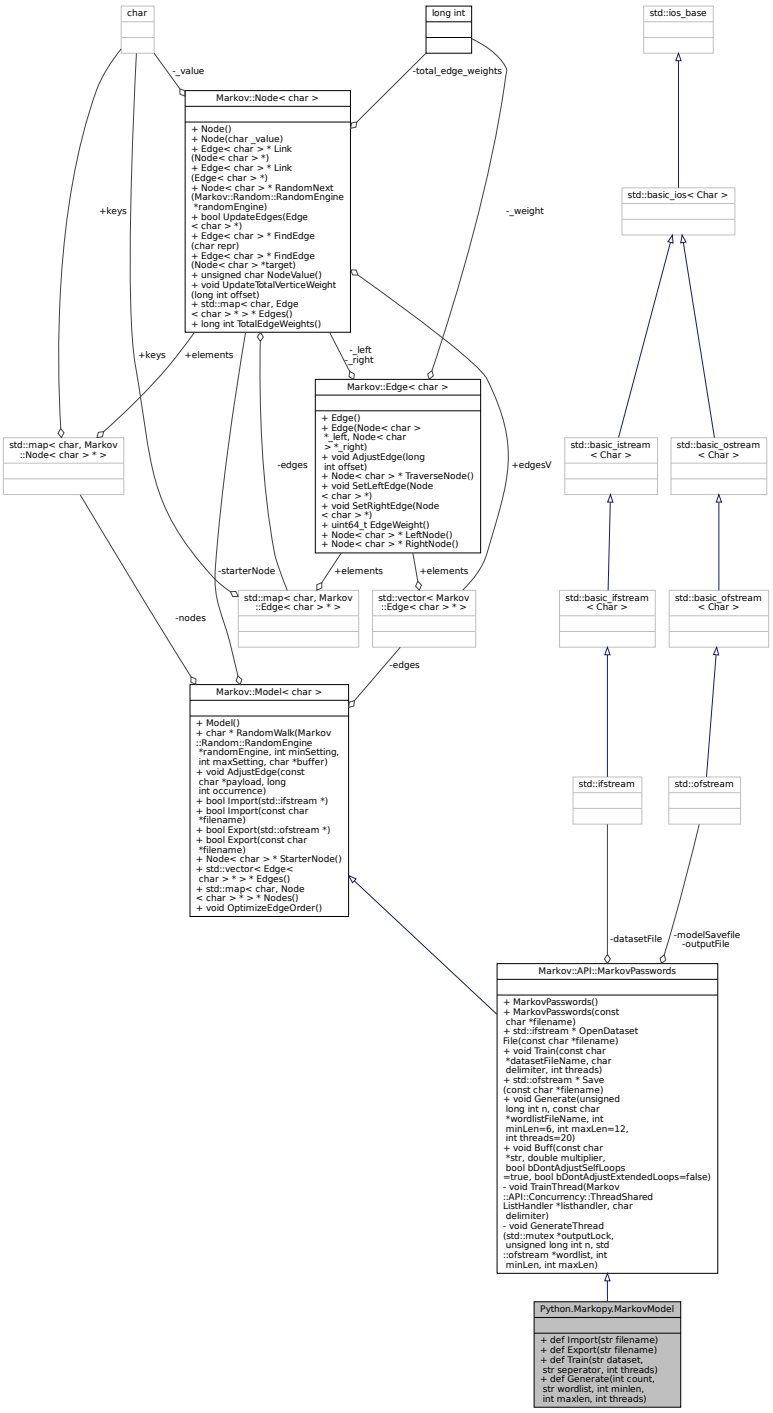
8.18 Python.Markopy.MarkovModel Class Reference

Abstract representation of a markov model.

Inheritance diagram for Python.Markopy.MarkovModel:



Collaboration diagram for Python.Markopy.MarkovModel:



Public Member Functions

- def Import (str filename)
- def Export (str filename)
- def Train (str dataset, str seperator, int threads)
- def Generate (int count, str wordlist, int minlen, int maxlen, int threads)
- std::ifstream * OpenDatasetFile (const char *filename)

  Open dataset file and return the ifstream pointer.

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

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

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

  Call Markov::Model::RandomWalk n times, and collect output.
- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- char * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

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

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

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

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

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

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

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

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

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.

Private Member Functions

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

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

  A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * datasetFile
- std::ofstream * modelSavefile

  Dataset file input of our system

- std::ofstream * outputFile

  File to save model of our system

- std::map< char, Node< char > * > nodes

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

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

  A list of all edges in this model.

8.18.1 Detailed Description

Abstract representation of a markov model.
To help with the python-cpp gateway documentation.
Definition at line 13 of file mm.py.

8.18.2 Member Function Documentation

8.18.2.1 AdjustEdge()

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

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

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

Parameters

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

Definition at line 109 of file model.h.

```
00337                                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.18.2.2 Buff()

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.

Parameters

bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

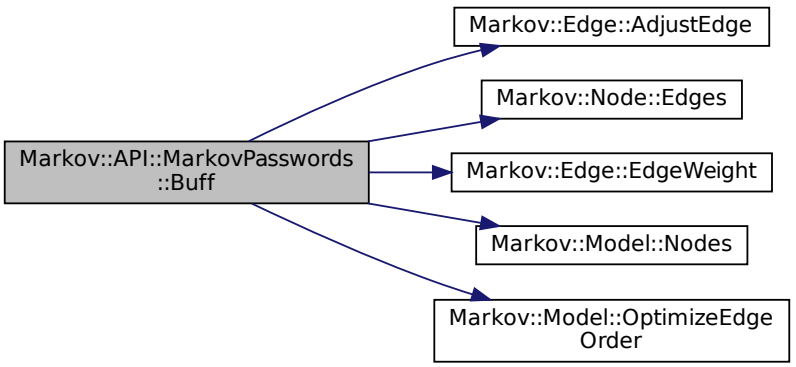
Definition at line 153 of file markovPasswords.cpp.

```
00153                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173 
00174         }
00175         i++;
00176     }
00177 
00178     this->OptimizeEdgeOrder();
00179 }
```

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

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.18.2.3 Edges()

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

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.18.2.4 Export() [1/3]

```
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 166 of file model.h.

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

8.18.2.5 Export() [2/3]

```
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 155 of file model.h.

```
00288                                                   {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
      e->RightNode()->NodeValue() « "\n";
00293         *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.18.2.6 Export() [3/3]

```
def Python.Markopy.MarkovModel.Export (
            str filename )
```

Definition at line 26 of file mm.py.

```
00026     def Export(filename : str):
00027         pass
00028
```

8.18.2.7 Generate() [1/2]

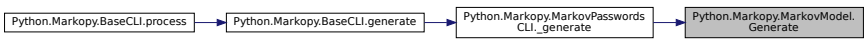
```
def Python.Markopy.MarkovModel.Generate (
            int count,
            str wordlist,
            int minlen,
            int maxlen,
            int threads )
```

Definition at line 34 of file mm.py.

```
00034     def Generate(count : int, wordlist : str, minlen : int, maxlen: int, threads : int):
00035         pass
00036
00037
```

Referenced by Python.Markopy.MarkovPasswordsCLI._generate().

Here is the caller graph for this function:



8.18.2.8 Generate() [2/2]

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

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

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

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

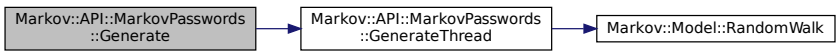
Parameters

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

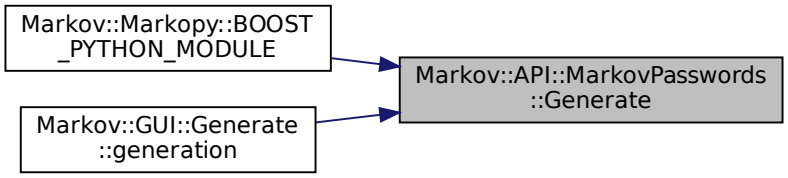
Definition at line 118 of file markovPasswords.cpp.

```
00118                                {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
      &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

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



Here is the caller graph for this function:



8.18.2.9 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
            std::mutex * outputLock,
```

```
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen,
            int maxLen )  [private], [inherited]
```

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

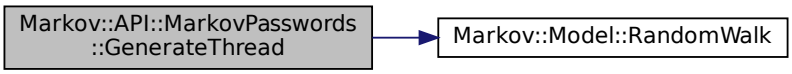
Parameters

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

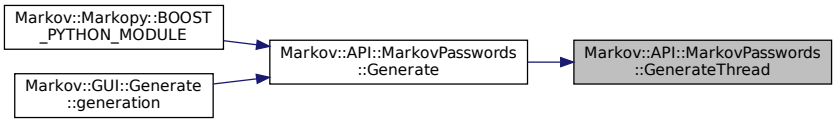
Definition at line 140 of file markovPasswords.cpp.

```
00140                                       {
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist « res « "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

References Markov::Model< NodeStorageType >::RandomWalk().
Referenced by Markov::API::MarkovPasswords::Generate().
Here is the call graph for this function:



Here is the caller graph for this function:



8.18.2.10 Import() [1/3]

```
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 137 of file model.h.

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

8.18.2.11 Import() [2/3]

```
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 126 of file model.h.

```
00216                                                  {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout « "cell: " « cell « std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout « oc « "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout « "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout « "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
```

```
00261     return true;
00262 }
```

8.18.2.12 Import() [3/3]

```
def Python.Markopy.MarkovModel.Import (
            str filename )
```

Definition at line 22 of file mm.py.

```
00022     def Import(filename : str):
00023         pass
00024
```

8.18.2.13 Nodes()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.18.2.14 OpenDatasetFile()

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

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

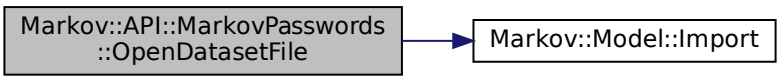
ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

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

Here is the call graph for this function:



8.18.2.15 OptimizeEdgeOrder()

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

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                 {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.18.2.16 RandomWalk()

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

Do a random walk on this model.

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

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

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

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

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

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

Parameters

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

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307                                                       {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
```

```
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319 
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323 
00324         n = temp_node;
00325 
00326         buffer[len++] = n->NodeValue();
00327     }
00328 
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331 
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.18.2.17 Save()

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

Export model to file.

Parameters

filename	- Export filename.

Returns

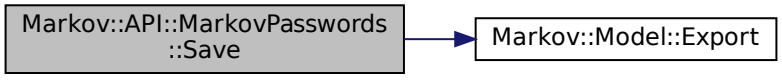
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                   {
00107     std::ofstream* exportFile;
00108 
00109     std::ofstream newFile(filename);
00110 
00111     exportFile = &newFile;
00112 
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

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

Here is the call graph for this function:



8.18.2.18 StarterNode()

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

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.18.2.19 Train() [1/2]

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

Train the model with the dataset file.

Parameters

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

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

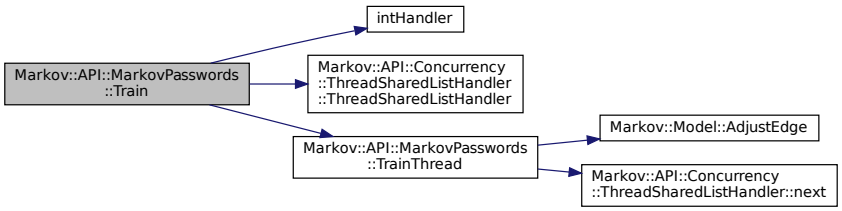
Definition at line 65 of file markovPasswords.cpp.

```
00065                                                                                                {
00066     signal(SIGINT, intHandler);
00067     Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
00068     auto start = std::chrono::high_resolution_clock::now();
00069
00070     std::vector<std::thread*> threadsV;
00071     for(int i=0;i<threads;i++){
00072         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
       &listhandler, delimiter));
00073     }
00074
00075     for(int i=0;i<threads;i++){
00076         threadsV[i]->join();
00077         delete threadsV[i];
00078     }
00079     auto finish = std::chrono::high_resolution_clock::now();
00080     std::chrono::duration<double> elapsed = finish - start;
00081     std::cout << "Elapsed time: " << elapsed.count() << " s\n";
00082
00083 }
```

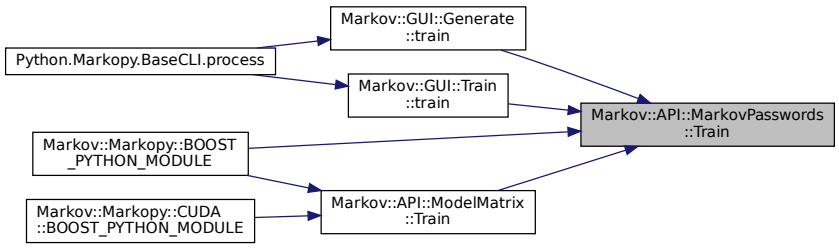
References intHandler(), Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler(), and Markov::API::MarkovPasswords::TrainThread().

Referenced by Markov::Markopy::BOOST_PYTHON_MODULE(), Markov::GUI::Generate::train(), Markov::GUI::Train::train(), and Markov::API::ModelMatrix::Train().

Here is the call graph for this function:



Here is the caller graph for this function:



8.18.2.20 Train() [2/2]

```
def Python.Markopy.MarkovModel.Train (
            str dataset,
            str seperator,
            int threads )
```

Definition at line 30 of file mm.py.

```
00030     def Train(dataset: str, seperator : str, threads : int):
00031         pass
00032
```

8.18.2.21 TrainThread()

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

A single thread invoked by the Train function.

Parameters

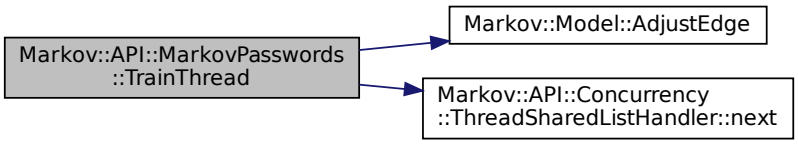
listhandler	- Listhandler class to read corpus from
delimiter	- a character, same as the delimiter in dataset content

Definition at line 85 of file markovPasswords.cpp.

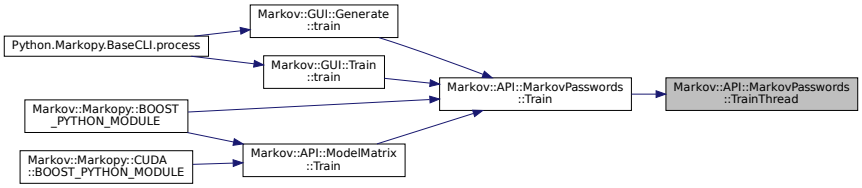
```
00085
                {
00086     char format_str[] ="%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
       "%ld,%s"
00097 #else
00098         sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100         this->AdjustEdge((const char*)linebuf, oc);
00101         delete linebuf;
00102     }
00103 }
```

References Markov::Model< NodeStorageType >::AdjustEdge(), keepRunning, and Markov::API::Concurrency::ThreadSharedListHa
Referenced by Markov::API::MarkovPasswords::Train().

Here is the call graph for this function:



Here is the caller graph for this function:



8.18.3 Member Data Documentation

8.18.3.1 datasetFile

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

8.18.3.2 edges

`std::vector<Edge<char >*> Markov::Model< char >::edges [private], [inherited]`
A list of all edges in this model.
Definition at line 204 of file model.h.

8.18.3.3 modelSavefile

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

Definition at line 124 of file markovPasswords.h.

8.18.3.4 nodes

`std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]`
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.

8.18.3.5 outputFile

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

Definition at line 125 of file markovPasswords.h.

8.18.3.6 starterNode

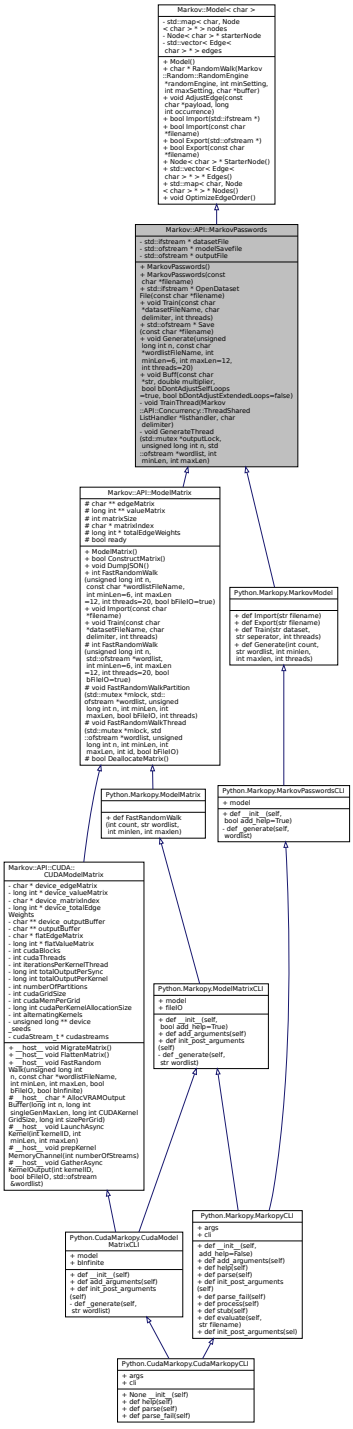
`Node<char >* Markov::Model< char >::starterNode [private], [inherited]`
Starter Node of this model.
Definition at line 198 of file model.h.
The documentation for this class was generated from the following file:

- Markopy/Markopy/src/CLI/mm.py

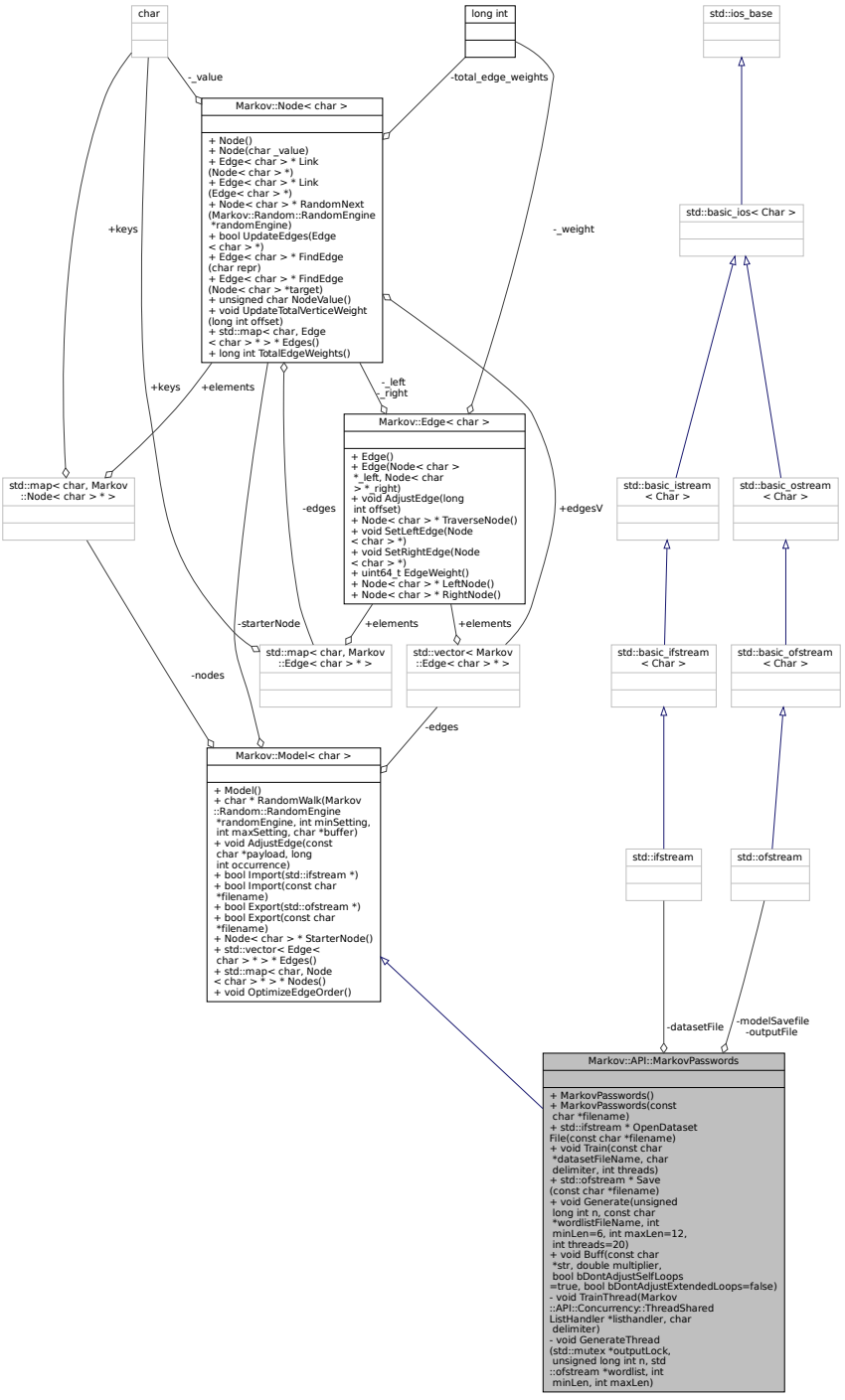
8.19 Markov::API::MarkovPasswords Class Reference

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

Inheritance diagram for Markov::API::MarkovPasswords:



Collaboration diagram for Markov::API::MarkovPasswords:



Public Member Functions

- MarkovPasswords ()

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

- MarkovPasswords (const char ∗filename)

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

- std::ifstream ∗ OpenDatasetFile (const char ∗filename)

  Open dataset file and return the ifstream pointer.

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

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

  Export model to file.
- void Generate (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

  Call Markov::Model::RandomWalk n times, and collect output.
- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- char * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

  Do a random walk on this model.
- void AdjustEdge (const char *payload, long int occurrence)

  Adjust the model with a single string.
- bool Import (std::ifstream *)

  Import a file to construct the model.
- bool Import (const char *filename)

  Open a file to import with filename, and call bool Model::Import with std::ifstream.
- bool Export (std::ofstream *)

  Export a file of the model.
- bool Export (const char *filename)

  Open a file to export with filename, and call bool Model::Export with std::ofstream.
- Node< char > * StarterNode ()

  Return starter Node.
- std::vector< Edge< char > * > * Edges ()

  Return a vector of all the edges in the model.
- std::map< char, Node< char > * > * Nodes ()

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.

Private Member Functions

- void TrainThread (Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char delimiter)

  A single thread invoked by the Train function.
- void GenerateThread (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)

  A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * datasetFile
- std::ofstream * modelSavefile

  Dataset file input of our system

- std::ofstream * outputFile

  File to save model of our system

- std::map< char, Node< char > * > nodes

  Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > * starterNode

  Starter Node of this model.
- std::vector< Edge< char > * > edges

  A list of all edges in this model.

8.19.1 Detailed Description

Markov::Model with char represented nodes.
Includes wrappers for Markov::Model and additional helper functions to handle file I/O
This class is an extension of Markov::Model<char>, with higher level abstractions such as train and generate.
Definition at line 26 of file markovPasswords.h.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 MarkovPasswords() [1/2]

```
Markov::API::MarkovPasswords::MarkovPasswords ( )
```

Initialize the markov model from MarkovModel::Markov::Model.
Parent constructor. Has no extra functionality.
Definition at line 34 of file markovPasswords.cpp.

```
00034                                           : Markov::Model<char>(){
00035
00036
00037 }
```

8.19.2.2 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

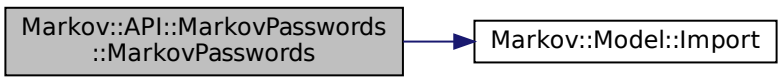
Example Use: Construction via filename

```
MarkovPasswords mp("test.mdl");
```

Definition at line 39 of file markovPasswords.cpp.

```
00039                                                                   {
00040
00041     std::ifstream* importFile;
00042
00043     this->Import(filename);
00044
00045     //std::ifstream* newFile(filename);
00046
00047     //importFile = newFile;
00048
00049 }
```

References Markov::Model< NodeStorageType >::Import().
Here is the call graph for this function:



8.19.3 Member Function Documentation

8.19.3.1 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
            const char * payload,
            long int occurrence )  [inherited]
```

Adjust the model with a single string.
Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.
Then, update the edge EdgeWeight from current node, to the terminator node.
This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.
Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file model.h.

```
00337                                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.19.3.2 Buff()

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file markovPasswords.cpp.

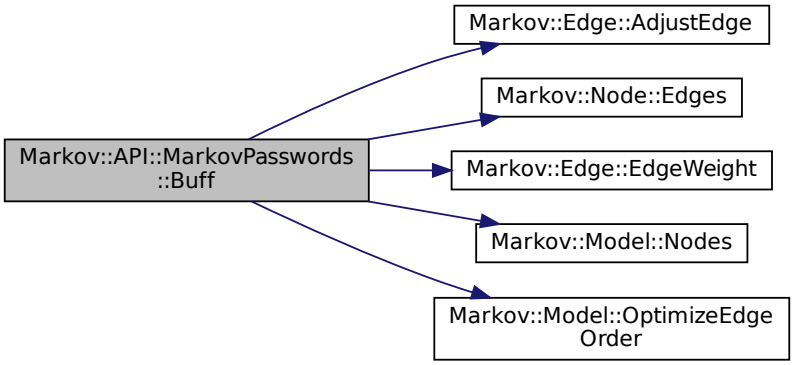
```
00153
                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
```

```
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173 
00174         }
00175         i++;
00176     }
00177 
00178     this->OptimizeEdgeOrder();
00179 }
```

References Markov::Edge< NodeStorageType >::AdjustEdge(), Markov::Node< storageType >::Edges(), Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::Model< NodeStorageType >::Nodes(), and Markov::Model< NodeStorageType >::OptimizeEdgeOrder().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.19.3.3 Edges()

```
std::vector<Edge<char >*>* Markov::Model< char >::Edges ( )  [inline], [inherited]
```

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.19.3.4 Export() [1/2]

```
bool Markov::Model< char >::Export (
            const char * filename )  [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file model.h.

```
00300                                                    {
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

8.19.3.5 Export() [2/2]

```
bool Markov::Model< char >::Export (
            std::ofstream * f )  [inherited]
```

Export a file of the model.
File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.
Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file model.h.

```
00288                                                {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
      e->RightNode()->NodeValue() << "\n";
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.19.3.6 Generate()

```
void Markov::API::MarkovPasswords::Generate (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20 )
```

Call Markov::Model::RandomWalk n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See Markov::API::MatrixModel::FastRandomWalk for more information.

Parameters

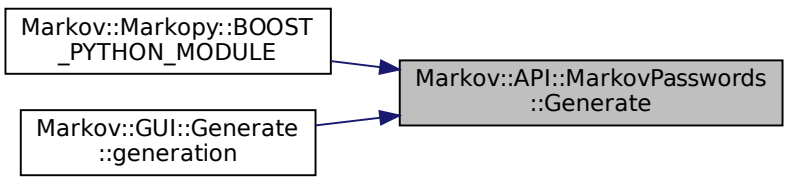
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

Definition at line 118 of file markovPasswords.cpp.

```
00118                                {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
      &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

References GenerateThread().
Referenced by Markov::Markopy::BOOST_PYTHON_MODULE(), and Markov::GUI::Generate::generation().
Here is the call graph for this function:

Here is the caller graph for this function:



8.19.3.7 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
            std::mutex * outputLock,
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen,
            int maxLen )  [private]
```

A single thread invoked by the Generate function.
DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

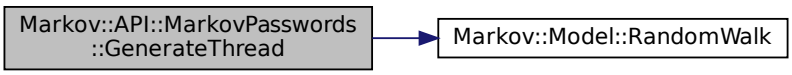
Parameters

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

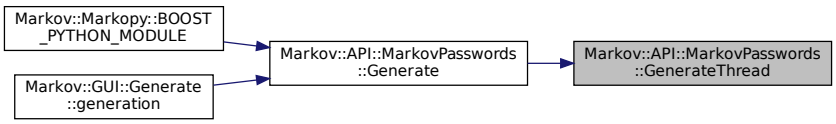
Definition at line 140 of file markovPasswords.cpp.

```
00140                                       {
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist << res << "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

References Markov::Model< NodeStorageType >::RandomWalk().
Referenced by Generate().
Here is the call graph for this function:



Here is the caller graph for this function:



8.19.3.8 Import() [1/2]

```
bool Markov::Model< char >::Import (
            const char * filename )  [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Definition at line 137 of file model.h.

```
00280                                                       {
00281     std::ifstream importfile;
00282     importfile.open(filename);
00283     return this->Import(&importfile);
00284
00285 }
```

8.19.3.9 Import() [2/2]

```
bool Markov::Model< char >::Import (
            std::ifstream * f )  [inherited]
```

Import a file to construct the model.
File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr
Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file model.h.

```
00216                                                   {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout << "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
```

```
00254         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

8.19.3.10 Nodes()

```
std::map<char , Node<char >*>* Markov::Model< char >::Nodes ( )  [inline], [inherited]
```

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.19.3.11 OpenDatasetFile()

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
            const char * filename )
```

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

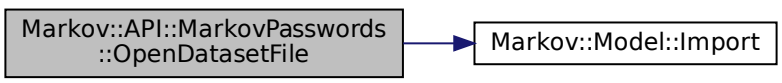
ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References Markov::Model< NodeStorageType >::Import().

Here is the call graph for this function:



8.19.3.12 OptimizeEdgeOrder()

```
void Markov::Model< char >::OptimizeEdgeOrder  [inherited]
```

Sort edges of all nodes in the model ordered by edge weights.
Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.19.3.13 RandomWalk()

```
char * Markov::Model< char >::RandomWalk (
            Markov::Random::RandomEngine * randomEngine,
            int minSetting,
            int maxSetting,
            char * buffer )  [inherited]
```

Do a random walk on this model.
Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length 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 86 of file model.h.

```
00307                                                        {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
```

```
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue();
00327     }
00328
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.19.3.14 Save()

```
std::ofstream * Markov::API::MarkovPasswords::Save (
            const char * filename )
```

Export model to file.

Parameters

filename	- Export filename.

Returns

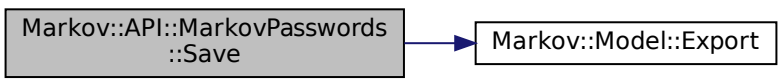
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                    {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References Markov::Model< NodeStorageType >::Export().

Here is the call graph for this function:



8.19.3.15 StarterNode()

```
Node<char >* Markov::Model< char >::StarterNode ( )  [inline], [inherited]
```

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.19.3.16 Train()

```
void Markov::API::MarkovPasswords::Train (
            const char * datasetFileName,
            char delimiter,
            int threads )
```

Train the model with the dataset file.

Parameters

datasetFileName	- Ifstream* to the dataset. If null, use class member
delimiter	- a character, same as the delimiter in dataset content
threads	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

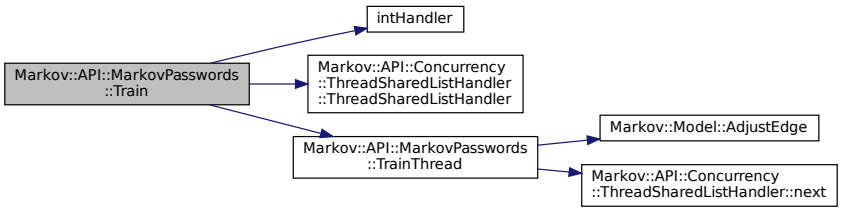
Definition at line 65 of file markovPasswords.cpp.

```
00065                                                                                                {
00066     signal(SIGINT, intHandler);
00067     Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
00068     auto start = std::chrono::high_resolution_clock::now();
00069
00070     std::vector<std::thread*> threadsV;
00071     for(int i=0;i<threads;i++){
00072         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
       &listhandler, delimiter));
00073     }
00074
00075     for(int i=0;i<threads;i++){
00076         threadsV[i]->join();
00077         delete threadsV[i];
00078     }
00079     auto finish = std::chrono::high_resolution_clock::now();
00080     std::chrono::duration<double> elapsed = finish - start;
00081     std::cout << "Elapsed time: " << elapsed.count() << " s\n";
00082
00083 }
```

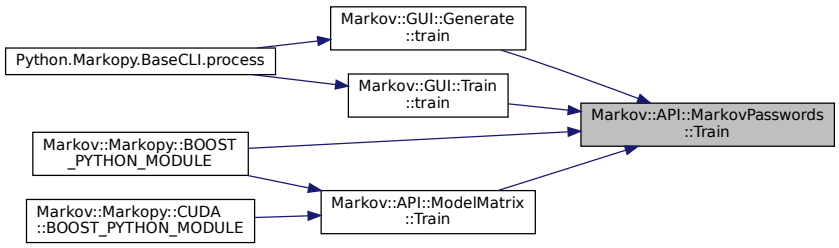
References intHandler(), Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler(), and TrainThread().

Referenced by Markov::Markopy::BOOST_PYTHON_MODULE(), Markov::GUI::Generate::train(), Markov::GUI::Train::train(), and Markov::API::ModelMatrix::Train().

Here is the call graph for this function:



Here is the caller graph for this function:



8.19.3.17 TrainThread()

```
void Markov::API::MarkovPasswords::TrainThread (
            Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
            char delimiter )  [private]
```

A single thread invoked by the Train function.

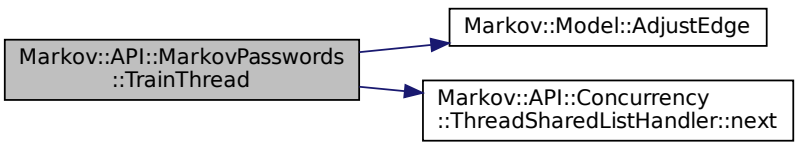
Parameters

listhandler	- Listhandler class to read corpus from
delimiter	- a character, same as the delimiter in dataset content

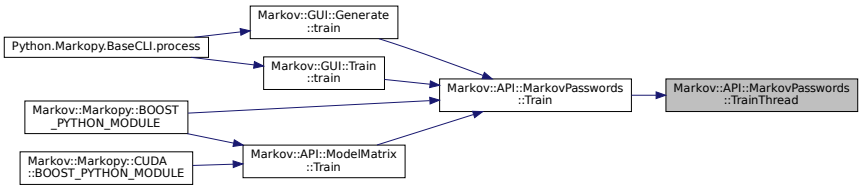
Definition at line 85 of file markovPasswords.cpp.

```
00085                 {
00086     char format_str[] ="%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
      "%ld,%s"
00097 #else
00098         sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100         this->AdjustEdge((const char*)linebuf, oc);
00101         delete linebuf;
00102     }
00103 }
```

References Markov::Model< NodeStorageType >::AdjustEdge(), keepRunning, and Markov::API::Concurrency::ThreadSharedListHa
Referenced by Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.19.4 Member Data Documentation

8.19.4.1 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile [private]`
Definition at line 123 of file markovPasswords.h.

8.19.4.2 edges

`std::vector<Edge<char >*> Markov::Model< char >::edges [private], [inherited]`
A list of all edges in this model.
Definition at line 204 of file model.h.

8.19.4.3 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private]`
Dataset file input of our system

Definition at line 124 of file markovPasswords.h.

8.19.4.4 nodes

`std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]`
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.

8.19.4.5 outputFile

`std::ofstream* Markov::API::MarkovPasswords::outputFile [private]`
File to save model of our system

Definition at line 125 of file markovPasswords.h.

8.19.4.6 starterNode

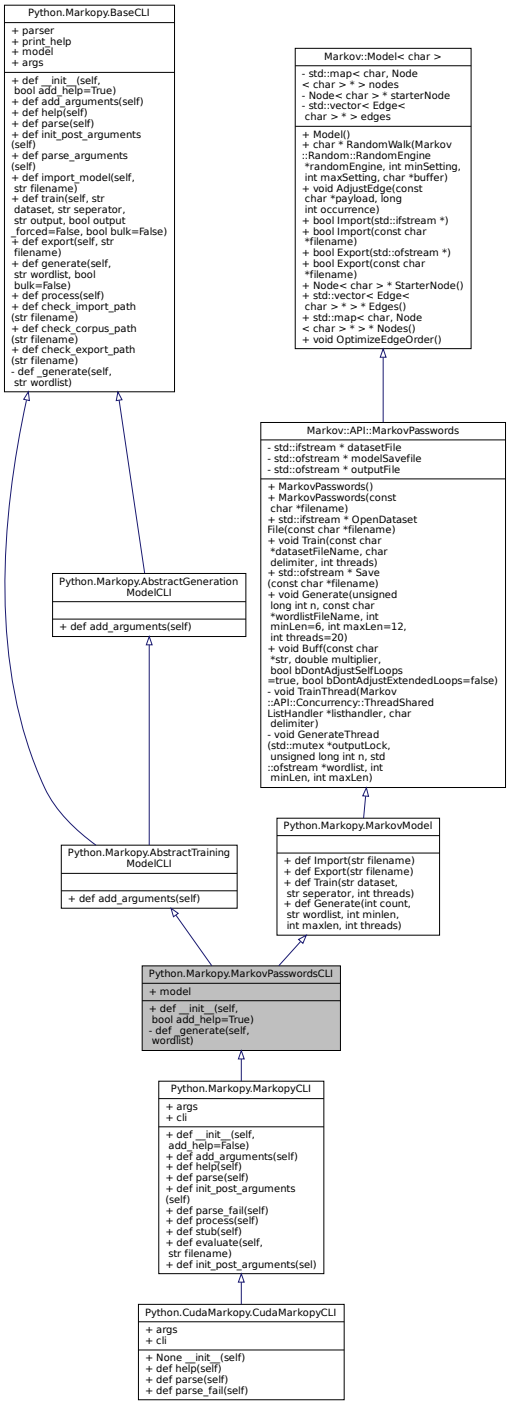
`Node<char >* Markov::Model< char >::starterNode [private], [inherited]`
Starter Node of this model.
Definition at line 198 of file model.h.
The documentation for this class was generated from the following files:

- Markopy/MarkovAPI/src/markovPasswords.h

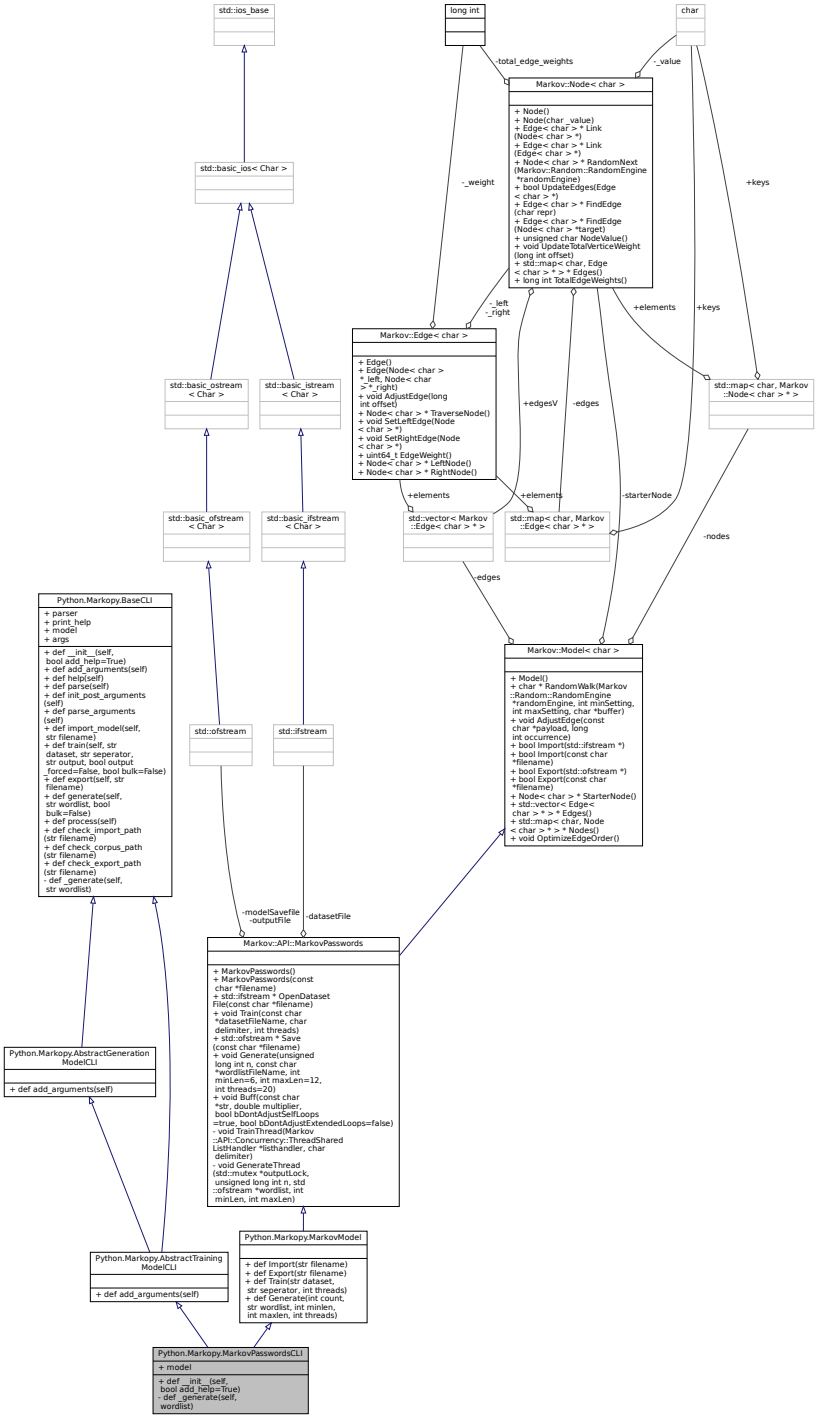
- Markopy/MarkovAPI/src/markovPasswords.cpp

8.20 Python.Markopy.MarkovPasswordsCLI Class Reference

Extension of Python.Markopy.Base.BaseCLI for Markov::API::MarkovPasswords.
Inheritance diagram for Python.Markopy.MarkovPasswordsCLI:



Collaboration diagram for Python.Markopy.MarkovPasswordsCLI:



Public Member Functions

- def __init__ (self, bool add_help=True)

  initialize base CLI
- def add_arguments (self)
- def help (self)
- def help (self)
- def parse (self)

- def parse (self)
- def init_post_arguments (self)
- def init_post_arguments (self)
- def parse_arguments (self)
- def parse_arguments (self)
- def import_model (self, str filename)

  Import a model file.
- def import_model (self, str filename)

  Import a model file.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def process (self)

  Process parameters for operation.
- def process (self)

  Process parameters for operation.
- def Import (str filename)
- 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.
- def Export (str filename)
- 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.
- def Train (str dataset, str seperator, int threads)
- void Train (const char *datasetFileName, char delimiter, int threads)

  Train the model with the dataset file.
- def Generate (int count, str wordlist, int minlen, int maxlen, int threads)
- 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.
- std::ifstream * OpenDatasetFile (const char *filename)

  Open dataset file and return the ifstream pointer.
- std::ofstream * Save (const char *filename)

  Export model to file.
- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- char * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

  Do a random walk on this model.

- void AdjustEdge (const char *payload, long int occurrence)

  Adjust the model with a single string.
- Node< char > * StarterNode ()

  Return starter Node.
- std::vector< Edge< char > * > * Edges ()

  Return a vector of all the edges in the model.
- std::map< char, Node< char > * > * Nodes ()

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.

Static Public Member Functions

- def check_import_path (str filename)

  check import path for validity
- def check_import_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity

Public Attributes

- model
- parser
- parser
- print_help
- print_help
- args
- args

Private Member Functions

- def _generate (self, wordlist)
- 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

  Dataset file input of our system

- std::ofstream * outputFile

  File to save model of our system

- std::map< char, Node< char > * > nodes

  Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > * starterNode

  Starter Node of this model.
- std::vector< Edge< char > * > edges

  A list of all edges in this model.

8.20.1 Detailed Description

Extension of Python.Markopy.Base.BaseCLI for Markov::API::MarkovPasswords.
adds -st/–stdout arguement to the command line.
Definition at line 17 of file mp.py.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 __init__()

```
def Python.Markopy.MarkovPasswordsCLI.__init__ (
              self,
             bool add_help = True )
```

initialize base CLI

Parameters

add_help	decide to overload the help function or not

Reimplemented from Python.Markopy.BaseCLI.
Definition at line 26 of file mp.py.

```
00026     def __init__(self, add_help:bool=True):
00027         "! @brief initialize model with Markov::API::MarkovPasswords"
00028         super().__init__(add_help)
00029         self.model = markopy.MarkovPasswords()
00030
```

8.20.3 Member Function Documentation

8.20.3.1 _generate()

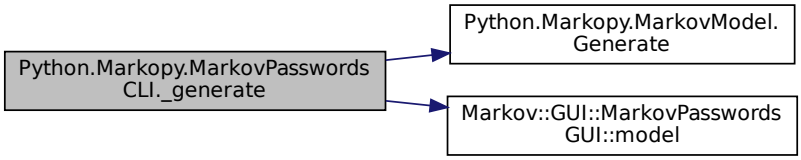
```
def Python.Markopy.MarkovPasswordsCLI._generate (
              self,
              wordlist )  [private]
```

Definition at line 31 of file mp.py.

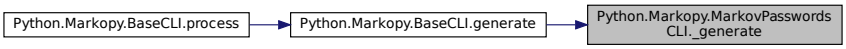
```
00031     def _generate(self, wordlist):
00032         "! @brief map generation function to Markov::API::MarkovPasswords::Generate"
00033         self.model.Generate(int(self.args.count), wordlist, int(self.args.min), int(self.args.max),
      int(self.args.threads))
00034
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.MarkovModel.Generate(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.mod
Referenced by Python.Markopy.BaseCLI.generate().

Here is the call graph for this function:



Here is the caller graph for this function:



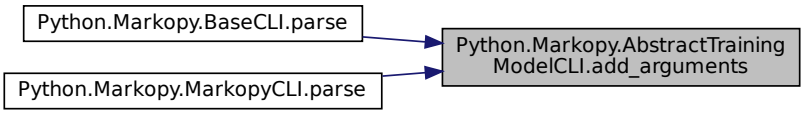
8.20.3.2 add_arguments()

```
def Python.Markopy.AbstractTrainingModelCLI.add_arguments (
              self )  [inherited]
```

Reimplemented from Python.Markopy.BaseCLI.
Reimplemented in Python.Markopy.MarkopyCLI.
Definition at line 282 of file base.py.

```
00282     def add_arguments(self):
00283         "Add command line arguements to the parser"
00284         self.parser.add_argument("-o", "--output",                  help="Output model file. This
      model will be exported when done. Will be ignored for generation mode.")
00285         self.parser.add_argument("-d", "--dataset",                 help="Dataset file to read input
      from for training. Will be ignored for generation mode.")
00286         self.parser.add_argument("-s", "--seperator",               help="Seperator character to use
      with training data.(character between occurrence and value)")
00287         super().add_arguments()
```

References Python.Markopy.BaseCLI.parser.
Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().
Here is the caller graph for this function:



8.20.3.3 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
             const NodeStorageType * payload,
             long int occurrence )  [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.
Then, update the edge EdgeWeight from current node, to the terminator node.
This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.
Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file model.h.

```
00337                                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.20.3.4 Buff()

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file markovPasswords.cpp.

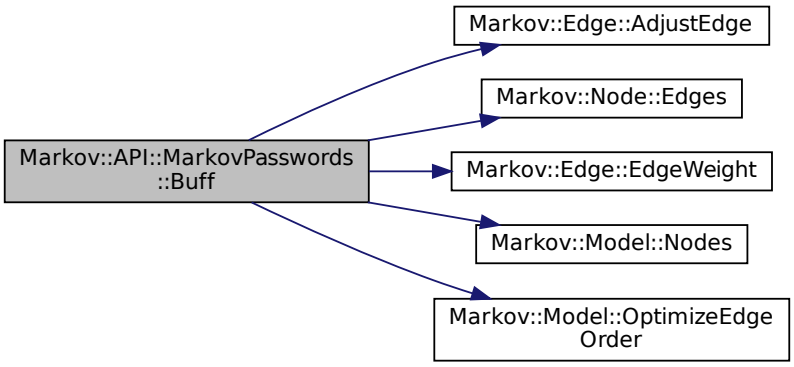
```
00153
                            {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
```

```
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173
00174         }
00175         i++;
00176     }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

References Markov::Edge< NodeStorageType >::AdjustEdge(), Markov::Node< storageType >::Edges(), Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::Model< NodeStorageType >::Nodes(), and Markov::Model< NodeStorageType >::OptimizeEdgeOrder().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.20.3.5 check_corpus_path() [1/2]

```
def Python.Markopy.BaseCLI.check_corpus_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186 
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.20.3.6 check_corpus_path() [2/2]

```
def Python.Markopy.BaseCLI.check_corpus_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186 
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.20.3.7 check_export_path() [1/2]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197 
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201 
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.20.3.8 check_export_path() [2/2]

```
def Python.Markopy.BaseCLI.check_export_path (
             str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197 
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201 
```

Referenced by Python.Markopy.BaseCLI.train().
Here is the caller graph for this function:

8.20.3.9 check_import_path() [1/2]

```
def Python.Markopy.BaseCLI.check_import_path (
             str filename )  [static], [inherited]
```

check import path for validity

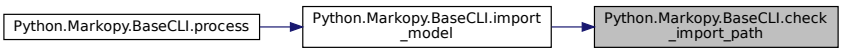
Parameters

filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:



8.20.3.10 check_import_path() [2/2]

```
def Python.Markopy.BaseCLI.check_import_path (
            str filename )  [static], [inherited]
```

check import path for validity

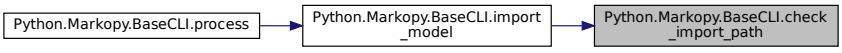
Parameters

filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
```

Referenced by Python.Markopy.BaseCLI.import_model().
Here is the caller graph for this function:



8.20.3.11 Edges()

```
std::vector<Edge<char >*>* Markov::Model< char >::Edges ( )  [inline], [inherited]
```

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.20.3.12 Export() [1/3]

```
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 166 of file model.h.

```
00300                                                          {
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

8.20.3.13 export() [1/2]

```
def Python.Markopy.BaseCLI.export (
             self,
            str filename )  [inherited]
```

Export model to a file.

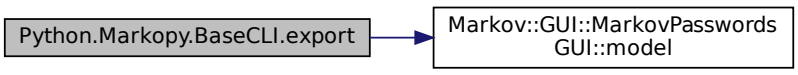
Parameters

filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.20.3.14 export() [2/2]

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

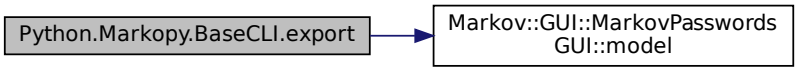
Parameters

filename	filename to export to

Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:

8.20.3.15 Export() [2/3]

```
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 155 of file model.h.

```
00288                                                {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
```

```
00292         //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
      e->RightNode()->NodeValue() « "\n";
00293         *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.20.3.16 Export() [3/3]

```
def Python.Markopy.MarkovModel.Export (
            str filename )  [inherited]
```

Definition at line 26 of file mm.py.

```
00026     def Export(filename : str):
00027         pass
00028
```

8.20.3.17 Generate() [1/2]

```
def Python.Markopy.MarkovModel.Generate (
            int count,
            str wordlist,
            int minlen,
            int maxlen,
            int threads )  [inherited]
```

Definition at line 34 of file mm.py.

```
00034     def Generate(count : int, wordlist : str, minlen : int, maxlen: int, threads : int):
00035         pass
00036
00037
```

Referenced by Python.Markopy.MarkovPasswordsCLI._generate().

Here is the caller graph for this function:

8.20.3.18 generate() [1/2]

```
def Python.Markopy.BaseCLI.generate (
             self,
            str wordlist,
            bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

Definition at line 145 of file base.py.

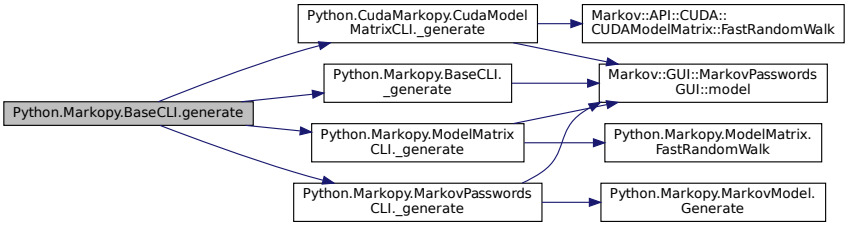
```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
```

```
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159 
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.20.3.19 generate() [2/2]

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

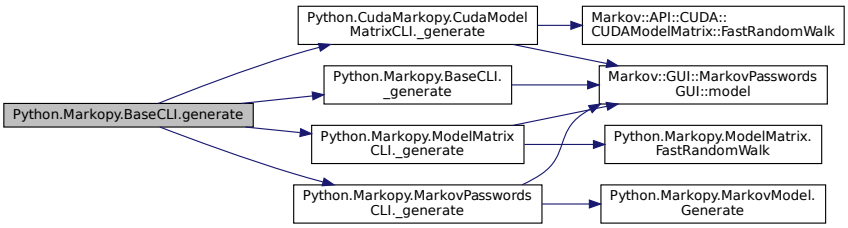
model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
```

```
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:

8.20.3.20 Generate() [2/2]

```
void Markov::API::MarkovPasswords::Generate (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20 )  [inherited]
```

Call Markov::Model::RandomWalk n times, and collect output.
Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See Markov::API::MatrixModel::FastRandomWalk for more information.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

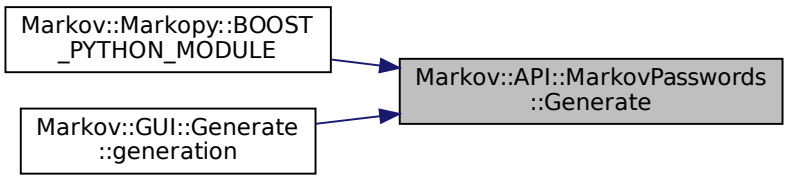
Definition at line 118 of file markovPasswords.cpp.

```
00118
                            {
00119     char* res;
```

```
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
      &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

References Markov::API::MarkovPasswords::GenerateThread().
Referenced by Markov::Markopy::BOOST_PYTHON_MODULE(), and Markov::GUI::Generate::generation().
Here is the call graph for this function:

Here is the caller graph for this function:



8.20.3.21 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
              std::mutex * outputLock,
              unsigned long int n,
              std::ofstream * wordlist,
              int minLen,
              int maxLen )  [private], [inherited]
```

A single thread invoked by the Generate function.
DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Parameters

outputLock	- shared mutex lock to lock during output operation. Prevents race condition on write.
n	number of lines to be generated by this thread
wordlist	wordlistfile

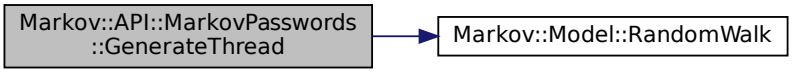
Parameters

minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

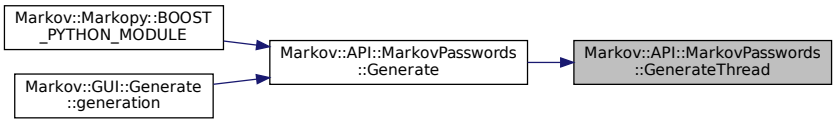
Definition at line 140 of file markovPasswords.cpp.

```
00140                                       {
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist « res « "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

References Markov::Model< NodeStorageType >::RandomWalk().
Referenced by Markov::API::MarkovPasswords::Generate().
Here is the call graph for this function:



Here is the caller graph for this function:



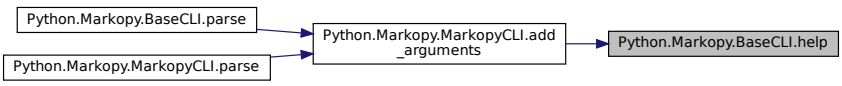
8.20.3.22 help() [1/2]

```
def Python.Markopy.BaseCLI.help (
              self )  [inherited]
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.
Definition at line 51 of file base.py.

```
00051     def help(self):
00052         "! @brief Handle help strings. Defaults to argparse's help"
00053         self.print_help()
00054
```

References Python.Markopy.BaseCLI.print_help.
Referenced by Python.Markopy.MarkopyCLI.add_arguments().
Here is the caller graph for this function:



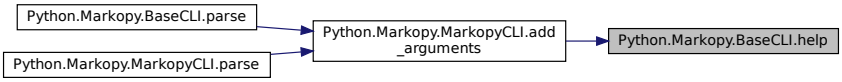
8.20.3.23 help() [2/2]

```
def Python.Markopy.BaseCLI.help (
              self )  [inherited]
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.
Definition at line 51 of file base.py.

```
00051     def help(self):
00052         "! @brief Handle help strings. Defaults to argparse's help"
00053         self.print_help()
00054
```

References Python.Markopy.BaseCLI.print_help.
Referenced by Python.Markopy.MarkopyCLI.add_arguments().
Here is the caller graph for this function:



8.20.3.24 Import() [1/3]

```
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 137 of file model.h.

```
00280                                                    {
00281     std::ifstream importfile;
00282     importfile.open(filename);
00283     return this->Import(&importfile);
00284
00285 }
```

8.20.3.25 Import() [2/3]

```
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 126 of file model.h.

```
00216                                                {
```

```
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout « "cell: " « cell « std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout « oc « "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout « "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout « "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

8.20.3.26 Import() [3/3]

```
def Python.Markopy.MarkovModel.Import (
            str filename )  [inherited]
```

Definition at line 22 of file mm.py.

```
00022     def Import(filename : str):
00023         pass
00024
```

8.20.3.27 import_model() [1/2]

```
def Python.Markopy.BaseCLI.import_model (
             self,
            str filename )  [inherited]
```

Import a model file.

Parameters

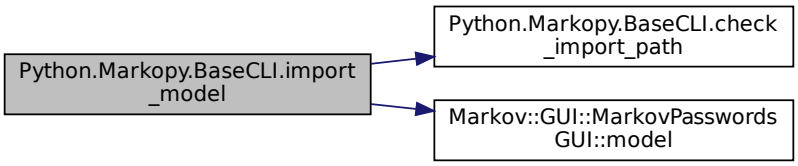
filename	filename to import

Definition at line 77 of file base.py.

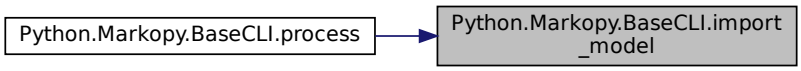
```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
```

```
00083
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



8.20.3.28 import_model() [2/2]

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

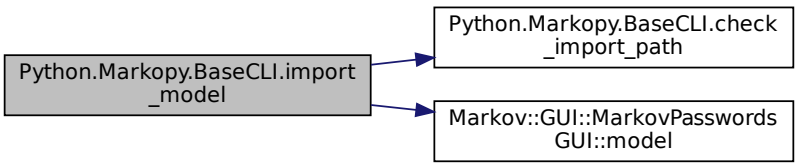
filename	filename to import

Definition at line 77 of file base.py.

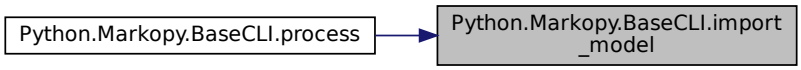
```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087
00088         self.model.Import(filename)
```

```
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



8.20.3.29 init_post_arguments() [1/2]

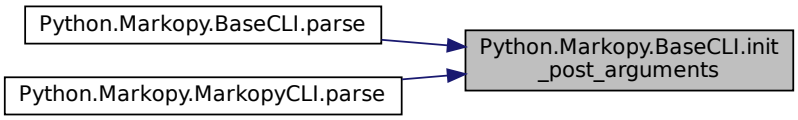
```
def Python.Markopy.BaseCLI.init_post_arguments (
              self )  [inherited]
```

Reimplemented in Python.Markopy.ModelMatrixCLI, Python.Markopy.MarkopyCLI, Python.CudaMarkopy.CudaModelMatrixCLI, and Python.Markopy.MarkopyCLI.
Definition at line 62 of file base.py.

```
00062     def init_post_arguments(self):
00063         "! @brief set up stuff that is collected from command line arguements"
00064         logging.VERBOSITY = 0
00065         try:
00066             if self.args.verbosity:
00067                 logging.VERBOSITY = self.args.verbosity
00068                 logging.pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00069         except:
00070             pass
00071
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.
Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.20.3.30 init_post_arguments() [2/2]

```
def Python.Markopy.BaseCLI.init_post_arguments (
              self )  [inherited]
```

Reimplemented in Python.Markopy.ModelMatrixCLI, Python.Markopy.MarkopyCLI, Python.CudaMarkopy.CudaModelMatrixCLI, and Python.Markopy.MarkopyCLI.

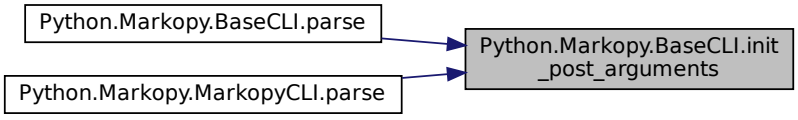
Definition at line 62 of file base.py.

```
00062     def init_post_arguments(self):
00063         "! @brief set up stuff that is collected from command line arguements"
00064         logging.VERBOSITY = 0
00065         try:
00066             if self.args.verbosity:
00067                 logging.VERBOSITY = self.args.verbosity
00068                 logging.pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00069         except:
00070             pass
00071
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.20.3.31 Nodes()

```
std::map<char , Node<char >*>* Markov::Model< char >::Nodes ( )  [inline], [inherited]
```

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.20.3.32 OpenDatasetFile()

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
             const char * filename )  [inherited]
```

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

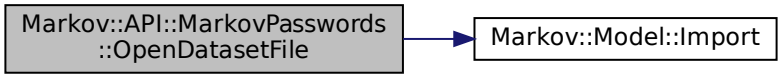
ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References Markov::Model< NodeStorageType >::Import().

Here is the call graph for this function:



8.20.3.33 OptimizeEdgeOrder()

```
void Markov::Model< char >::OptimizeEdgeOrder  [inherited]
```

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.20.3.34 parse() [1/2]

```
def Python.Markopy.BaseCLI.parse (
              self )  [inherited]
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.

Definition at line 55 of file base.py.

```
00055     def parse(self):
00056         "! @brief add, parse and hook arguements"
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(),

Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMode
Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.init_post_arguments(), Python.Markopy.ModelMatrixC
and Python.Markopy.BaseCLI.parse_arguments().
Here is the call graph for this function:

8.20.3.35 parse() [2/2]

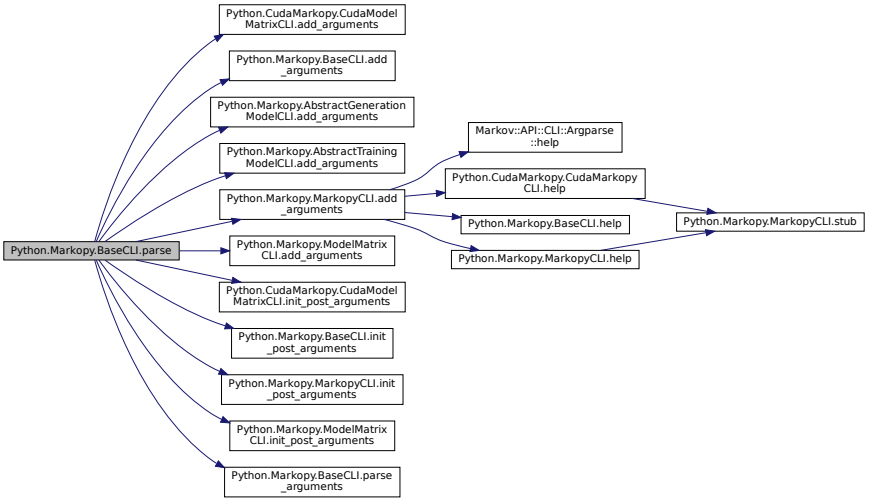
```
def Python.Markopy.BaseCLI.parse (
              self )  [inherited]
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.
Definition at line 55 of file base.py.

```
00055     def parse(self):
00056         "! @brief add, parse and hook arguements"
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMode
Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.init_post_arguments(), Python.Markopy.ModelMatrixC
and Python.Markopy.BaseCLI.parse_arguments().

Here is the call graph for this function:



8.20.3.36 parse_arguments() [1/2]

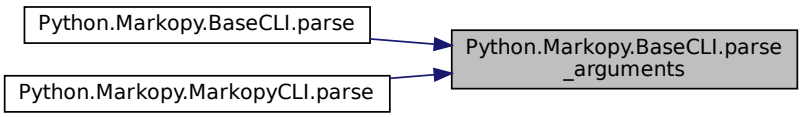
```
def Python.Markopy.BaseCLI.parse_arguments (
            self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.20.3.37 parse_arguments() [2/2]

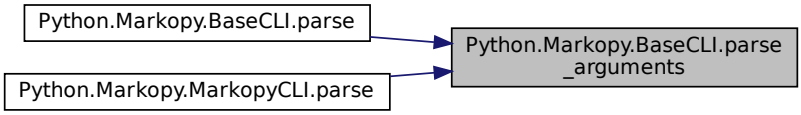
```
def Python.Markopy.BaseCLI.parse_arguments (
            self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.20.3.38 process() [1/2]

```
def Python.Markopy.BaseCLI.process (
            self )  [inherited]
```

Process parameters for operation.

Reimplemented in Python.Markopy.MarkopyCLI.

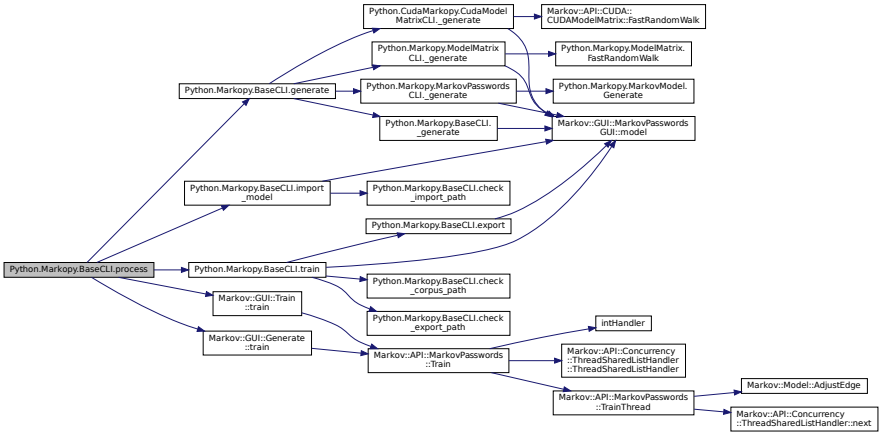
Definition at line 202 of file base.py.

```
00202     def process(self):
00203         """!
00204         @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging.pprint(f"Bulk mode operation chosen.", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
      (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00210                     corpus_list = os.listdir(self.args.dataset)
00211                     for corpus in corpus_list:
00212                         self.import_model(self.args.input)
00213                         logging.pprint(f"Training {self.args.input} with {corpus}", 2)
00214                         output_file_name = corpus
00215                         model_extension = ""
00216                         if "." in self.args.input:
00217                             model_extension = self.args.input.split(".")[-1]
00218                         self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
      f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00219                 else:
00220                     logging.pprint("In bulk training, output and dataset should be a directory.")
00221                     exit(1)
00222
00223             elif (self.args.mode.lower() == "generate"):
00224                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
      (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00225                     model_list = os.listdir(self.args.input)
00226                     print(model_list)
00227                     for input in model_list:
00228                         logging.pprint(f"Generating from {self.args.input}/{input} to
      {self.args.wordlist}/{input}.txt", 2)
00229                         self.import_model(f"{self.args.input}/{input}")
00230                         model_base = input
00231                         if "." in self.args.input:
00232                             model_base = input.split(".")[1]
00233                         self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00234                 else:
00235                     logging.pprint("In bulk generation, input and wordlist should be directory.")
00236
00237         else:
00238             self.import_model(self.args.input)
00239             if (self.args.mode.lower() == "generate"):
00240                 self.generate(self.args.wordlist)
00241
00242
00243             elif (self.args.mode.lower() == "train"):
00244                 self.train(self.args.dataset, self.args.seperator, self.args.output,
      output_forced=True)
00245
00246
00247             elif(self.args.mode.lower() == "combine"):
00248                 self.train(self.args.dataset, self.args.seperator, self.args.output)
00249                 self.generate(self.args.wordlist)
00250
00251
00252             else:
00253                 logging.pprint("Invalid mode arguement given.")
```

```
00254                 logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255                 exit(5)
00256
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.generate(), Python.Markopy.BaseCLI.import_model(), Markov::GUI::Generate.train(), Markov::GUI::Train.train(), and Python.Markopy.BaseCLI.train().

Here is the call graph for this function:



8.20.3.39 process() [2/2]

```
def Python.Markopy.BaseCLI.process (
            self )  [inherited]
```

Process parameters for operation.

Reimplemented in Python.Markopy.MarkopyCLI.

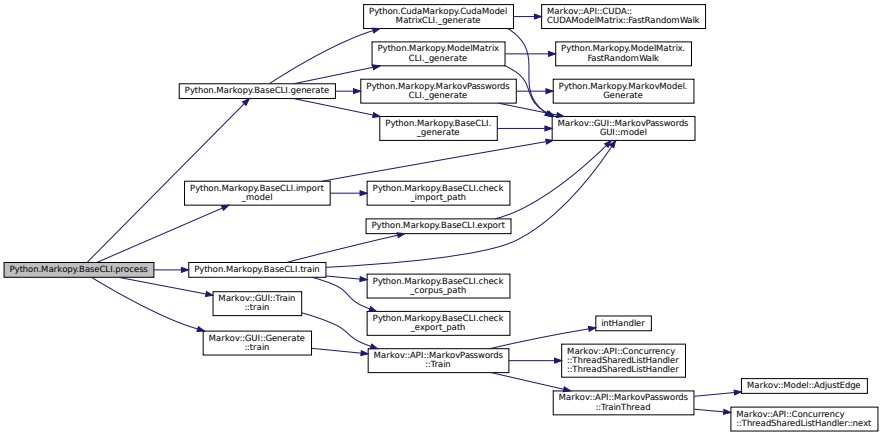
Definition at line 202 of file base.py.

```
00202     def process(self):
00203         """!
00204         @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging.pprint(f"Bulk mode operation chosen.", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
      (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00210                     corpus_list = os.listdir(self.args.dataset)
00211                     for corpus in corpus_list:
00212                         self.import_model(self.args.input)
00213                         logging.pprint(f"Training {self.args.input} with {corpus}", 2)
00214                         output_file_name = corpus
00215                         model_extension = ""
00216                         if "." in self.args.input:
00217                             model_extension = self.args.input.split(".")[-1]
00218                         self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
      f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00219                 else:
00220                     logging.pprint("In bulk training, output and dataset should be a directory.")
00221                     exit(1)
00222
00223             elif (self.args.mode.lower() == "generate"):
00224                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
      (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00225                     model_list = os.listdir(self.args.input)
00226                     print(model_list)
00227                     for input in model_list:
00228                         logging.pprint(f"Generating from {self.args.input}/{input} to
      {self.args.wordlist}/{input}.txt", 2)
00229                         self.import_model(f"{self.args.input}/{input}")
00230                         model_base = input
00231                         if "." in self.args.input:
00232                             model_base = input.split(".")[1]
00233                         self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00234                 else:
00235                     logging.pprint("In bulk generation, input and wordlist should be directory.")
```

```
00236
00237         else:
00238             self.import_model(self.args.input)
00239             if (self.args.mode.lower() == "generate"):
00240                 self.generate(self.args.wordlist)
00241
00242
00243             elif (self.args.mode.lower() == "train"):
00244                 self.train(self.args.dataset, self.args.seperator, self.args.output,
      output_forced=True)
00245
00246
00247             elif(self.args.mode.lower() == "combine"):
00248                 self.train(self.args.dataset, self.args.seperator, self.args.output)
00249                 self.generate(self.args.wordlist)
00250
00251
00252             else:
00253                 logging.pprint("Invalid mode arguement given.")
00254                 logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255                 exit(5)
00256
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.generate(), Python.Markopy.BaseCLI.import_model(), Markov::GUI::Generate.train(), Markov::GUI::Train.train(), and Python.Markopy.BaseCLI.train().

Here is the call graph for this function:



8.20.3.40 RandomWalk()

```
char * Markov::Model< char >::RandomWalk (
            Markov::Random::RandomEngine * randomEngine,
            int minSetting,
            int maxSetting,
            NodeStorageType * buffer )  [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criateria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes Markov::Random::RandomEngine as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so Markov::Random::Marsaglia is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
```

```
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout « res « "\n";
 }
```

Parameters

randomEngine	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
minSetting	Minimum number of characters to generate
maxSetting	Maximum number of character to generate
buffer	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307                                                                {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue();
00327     }
00328
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.20.3.41 Save()

```
std::ofstream * Markov::API::MarkovPasswords::Save (
            const char * filename )  [inherited]
```

Export model to file.

Parameters

filename	- Export filename.

Returns

std::ofstream* of the exported file.

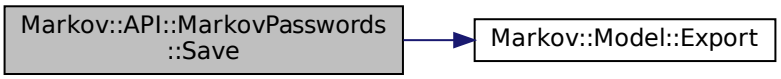
Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
```

```
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References Markov::Model< NodeStorageType >::Export().

Here is the call graph for this function:



8.20.3.42 StarterNode()

```
Node<char >* Markov::Model< char >::StarterNode ( )  [inline], [inherited]
```

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.20.3.43 Train() [1/2]

```
void Markov::API::MarkovPasswords::Train (
            const char * datasetFileName,
            char delimiter,
            int threads )  [inherited]
```

Train the model with the dataset file.

Parameters

datasetFileName	- Ifstream* to the dataset. If null, use class member
delimiter	- a character, same as the delimiter in dataset content
threads	- number of OS threads to spawn

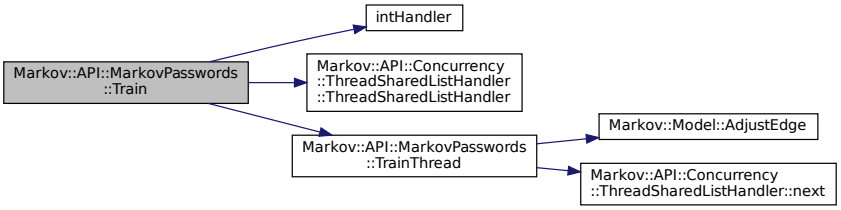
```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

Definition at line 65 of file markovPasswords.cpp.

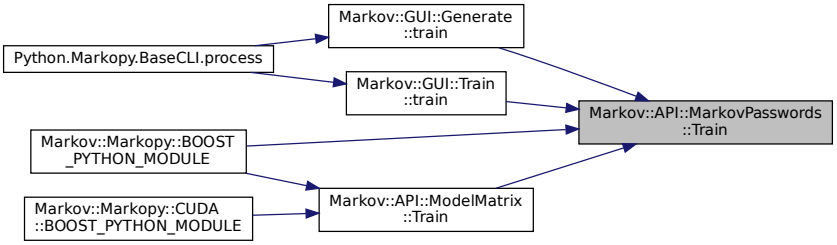
```
00065                                                                                                {
00066     signal(SIGINT, intHandler);
00067     Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
00068     auto start = std::chrono::high_resolution_clock::now();
00069
00070     std::vector<std::thread*> threadsV;
00071     for(int i=0;i<threads;i++){
00072         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
       &listhandler, delimiter));
00073     }
00074
00075     for(int i=0;i<threads;i++){
00076         threadsV[i]->join();
00077         delete threadsV[i];
00078     }
00079     auto finish = std::chrono::high_resolution_clock::now();
00080     std::chrono::duration<double> elapsed = finish - start;
```

```
00081     std::cout « "Elapsed time: " « elapsed.count() « " s\n";
00082
00083 }
```

References intHandler(), Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler(), and Markov::API::MarkovPasswords::TrainThread().
Referenced by Markov::Markopy::BOOST_PYTHON_MODULE(), Markov::GUI::Generate::train(), Markov::GUI::Train::train(), and Markov::API::ModelMatrix::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.20.3.44 train() [1/2]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

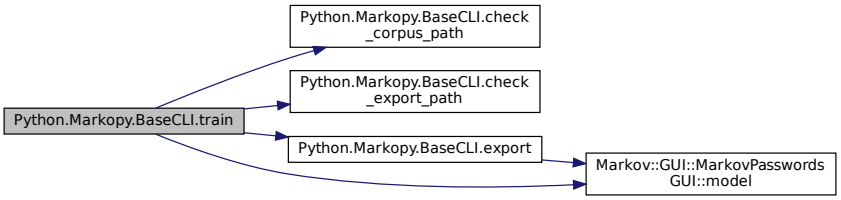
Definition at line 94 of file base.py.

```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

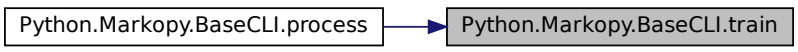
References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().

Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.20.3.45 train() [2/2]

```
def Python.Markopy.BaseCLI.train (
              self,
             str dataset,
             str seperator,
             str output,
             bool output_forced = False,
             bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

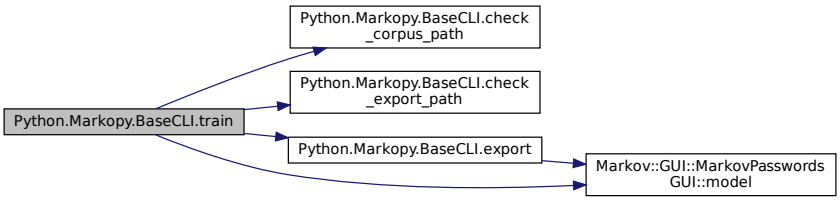
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

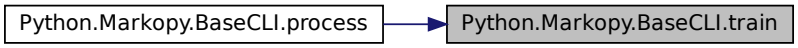
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model,

Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



8.20.3.46 Train() [2/2]

```
def Python.Markopy.MarkovModel.Train (
            str dataset,
            str seperator,
            int threads )  [inherited]
```

Definition at line 30 of file mm.py.

```
00030     def Train(dataset: str, seperator : str, threads : int):
00031         pass
00032
```

8.20.3.47 TrainThread()

```
void Markov::API::MarkovPasswords::TrainThread (
            Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
            char delimiter )  [private], [inherited]
```

A single thread invoked by the Train function.

Parameters

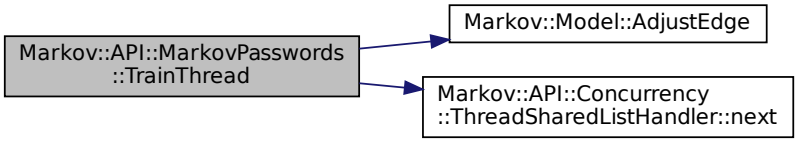
listhandler	- Listhandler class to read corpus from
delimiter	- a character, same as the delimiter in dataset content

Definition at line 85 of file markovPasswords.cpp.

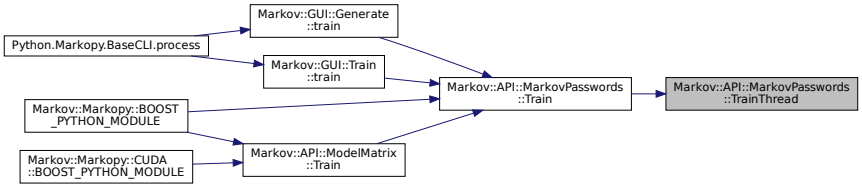
```
00085
                {
00086     char format_str[] ="%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
```

```
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
      "%ld,%s"
00097 #else
00098         sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100         this->AdjustEdge((const char*)linebuf, oc);
00101         delete linebuf;
00102     }
00103 }
```

References Markov::Model< NodeStorageType >::AdjustEdge(), keepRunning, and Markov::API::Concurrency::ThreadSharedListHa
Referenced by Markov::API::MarkovPasswords::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.20.4 Member Data Documentation

8.20.4.1 args [1/2]

Python.Markopy.BaseCLI.args [inherited]
Definition at line 75 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.generate Python.Markopy.MarkopyCLI.help(), Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.parse(), Python.CudaMarkopy.CudaMarkopyCLI.parse_fail(), Python.Markopy.BaseCLI.process(), and Python.Markopy.BaseCLI.train().

8.20.4.2 args [2/2]

Python.Markopy.BaseCLI.args [inherited]
Definition at line 75 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.generate Python.Markopy.MarkopyCLI.help(), Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.parse(), Python.CudaMarkopy.CudaMarkopyCLI.parse_fail(), Python.Markopy.BaseCLI.process(), and Python.Markopy.BaseCLI.train().

8.20.4.3 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]`
Definition at line 123 of file markovPasswords.h.

8.20.4.4 edges

`std::vector<Edge<char >*> Markov::Model< char >::edges [private], [inherited]`
A list of all edges in this model.
Definition at line 204 of file model.h.

8.20.4.5 model

`Python.Markopy.MarkovPasswordsCLI.model`
Definition at line 29 of file mp.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.20.4.6 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]`
Dataset file input of our system

Definition at line 124 of file markovPasswords.h.

8.20.4.7 nodes

`std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]`
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.

8.20.4.8 outputFile

`std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]`
File to save model of our system

Definition at line 125 of file markovPasswords.h.

8.20.4.9 parser [1/2]

`Python.Markopy.BaseCLI.parser [inherited]`
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.20.4.10 parser [2/2]

`Python.Markopy.BaseCLI.parser [inherited]`
Definition at line 25 of file base.py.

Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.20.4.11 print_help [1/2]

Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.20.4.12 print_help [2/2]

Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.20.4.13 starterNode

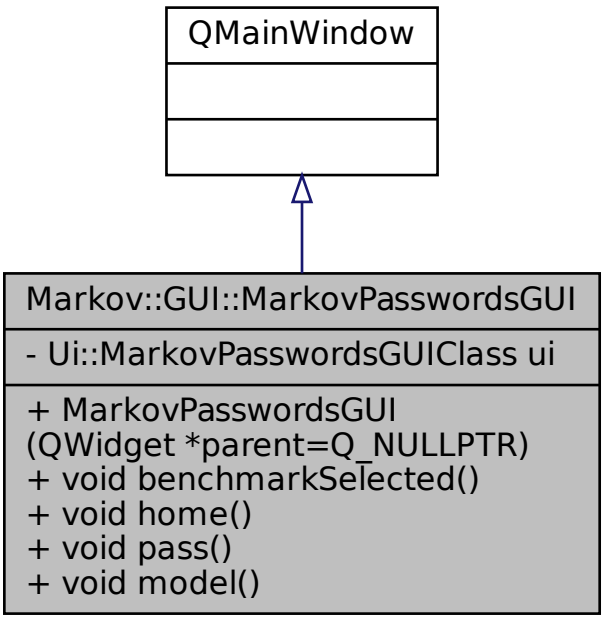
Node<char >* Markov::Model< char >::starterNode [private], [inherited]
Starter Node of this model.
Definition at line 198 of file model.h.
The documentation for this class was generated from the following file:

- Markopy/Markopy/src/CLI/mp.py

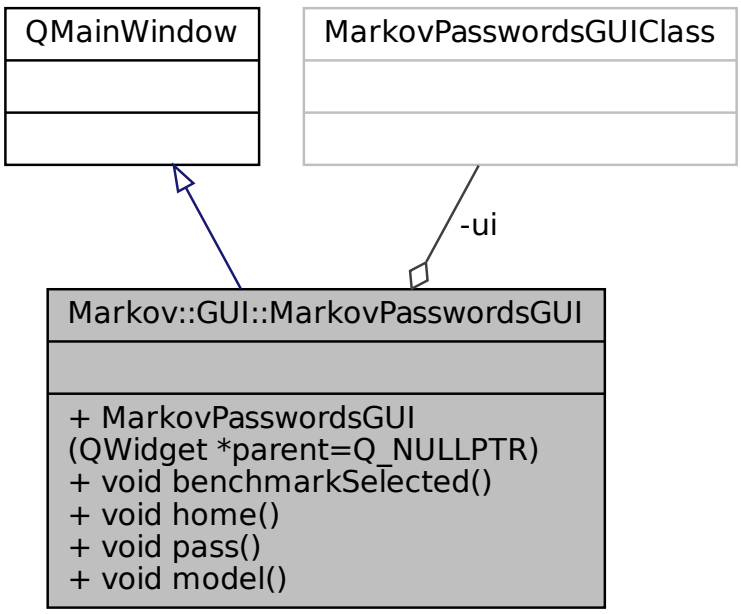
8.21 Markov::GUI::MarkovPasswordsGUI Class Reference

Reporting UI.
#include <MarkovPasswordsGUI.h>

Inheritance diagram for Markov::GUI::MarkovPasswordsGUI:



Collaboration diagram for Markov::GUI::MarkovPasswordsGUI:



Public Slots

- void benchmarkSelected ()
- void home ()
- void pass ()
- void model ()

Public Member Functions

- MarkovPasswordsGUI (QWidget *parent=Q_NULLPTR)

Private Attributes

- Ui::MarkovPasswordsGUIClass ui

8.21.1 Detailed Description

Reporting UI.
UI for reporting and debugging tools for MarkovPassword
Definition at line 19 of file MarkovPasswordsGUI.h.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 MarkovPasswordsGUI()

```
Markov::GUI::MarkovPasswordsGUI::MarkovPasswordsGUI (
            QWidget * parent = Q_NULLPTR )
```

Definition at line 14 of file MarkovPasswordsGUI.cpp.

```
00014                                                       : QMainWindow(parent){
00015     ui.setupUi(this);
00016
00017
00018     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {home(); });
00019     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {model(); });
00020     QObject::connect(ui.pushButton_3, &QPushButton::clicked, this, [this] {pass(); });
00021 }
```

References ui.

8.21.3 Member Function Documentation

8.21.3.1 benchmarkSelected

```
void Markov::GUI::MarkovPasswordsGUI::benchmarkSelected ( )  [slot]
```

8.21.3.2 home

```
void MarkovPasswordsGUI::home ( )  [slot]
```

Definition at line 24 of file MarkovPasswordsGUI.cpp.

```
00024                                {
00025     CLI* w = new CLI;
00026     w->show();
00027     this->close();
00028 }
```

8.21.3.3 model

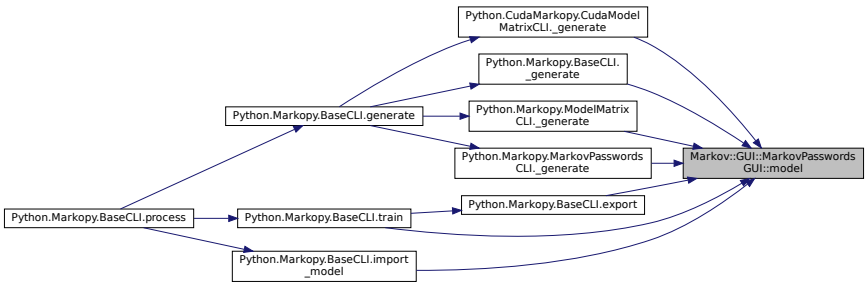
```
void MarkovPasswordsGUI::model ( )  [slot]
```

Definition at line 42 of file MarkovPasswordsGUI.cpp.

```
00042                                {
00043     QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00044
00045     //get working directory
00046     char path[255];
00047     GetCurrentDirectoryA(255, path);
00048
00049     //get absolute path to the layout html
00050     std::string layout = "file:///" + std::string(path) + "\\views\\index.html";
00051     std::replace(layout.begin(), layout.end(), '\\', '/');
00052     webkit->setUrl(QUrl(layout.c_str()));
00053 }
```

Referenced by Python.CudaMarkopy.CudaModelMatrixCLI::_generate(), Python.Markopy.BaseCLI::_generate(), Python.Markopy.ModelMatrixCLI::_generate(), Python.Markopy.MarkovPasswordsCLI::_generate(), Python.Markopy.BaseCLI::export(), Python.Markopy.BaseCLI::import_model(), and Python.Markopy.BaseCLI::train().

Here is the caller graph for this function:



8.21.3.4 pass

```
void MarkovPasswordsGUI::pass ( )  [slot]
```

Definition at line 29 of file MarkovPasswordsGUI.cpp.

```
00029                                {
00030     QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00031
00032     //get working directory
00033     char path[255];
00034     GetCurrentDirectoryA(255, path);
00035
00036     //get absolute path to the layout html
00037     std::string layout = "file:///" + std::string(path) + "\\views\\bar.html";
00038     std::replace(layout.begin(), layout.end(), '\\', '/');
00039     webkit->setUrl(QUrl(layout.c_str()));
00040 }
```

8.21.4 Member Data Documentation

8.21.4.1 ui

```
Ui::MarkovPasswordsGUIClass Markov::GUI::MarkovPasswordsGUI::ui  [private]
```

Definition at line 25 of file MarkovPasswordsGUI.h.

Referenced by MarkovPasswordsGUI().

The documentation for this class was generated from the following files:

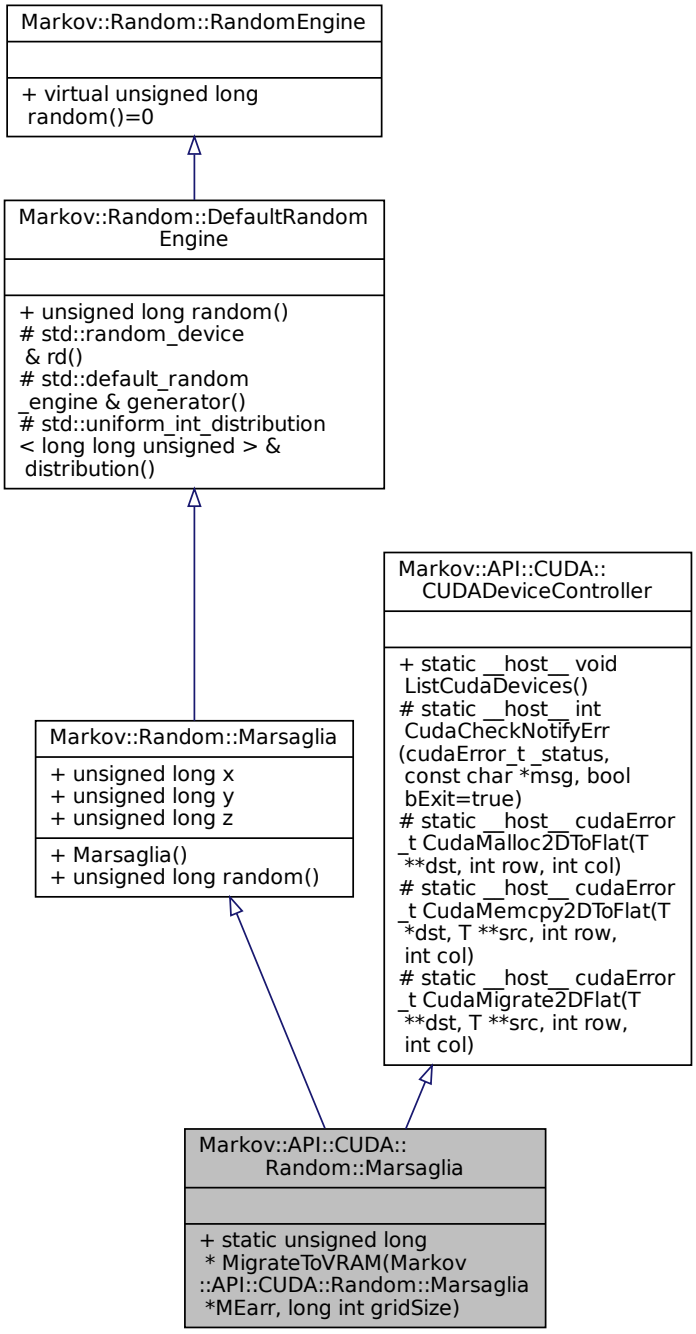
- Markopy/MarkovPasswordsGUI/src/MarkovPasswordsGUI.h
- Markopy/MarkovPasswordsGUI/src/MarkovPasswordsGUI.cpp

8.22 Markov::API::CUDA::Random::Marsaglia Class Reference

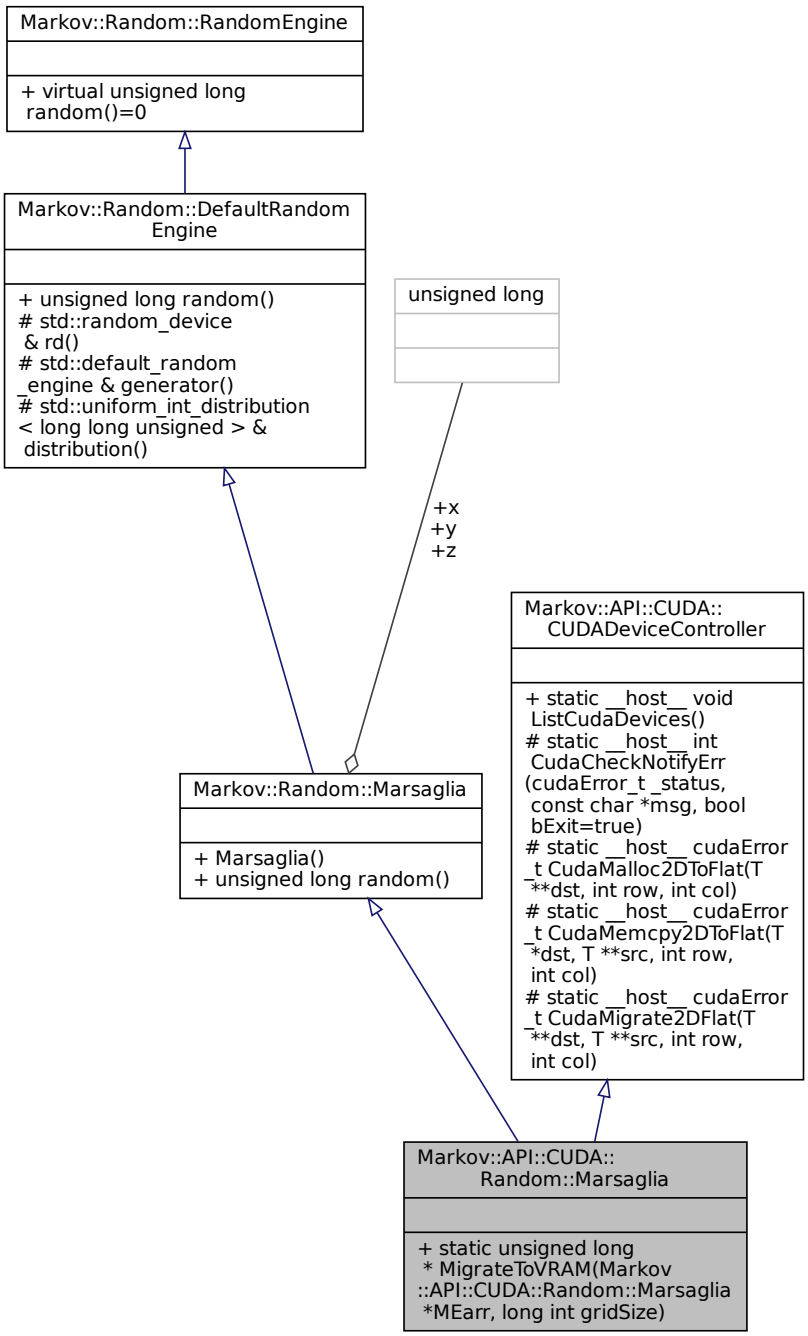
Extension of Markov::Random::Marsaglia which is capable o working on **device** space.

`#include <cudarandom.h>`

Inheritance diagram for Markov::API::CUDA::Random::Marsaglia:



Collaboration diagram for Markov::API::CUDA::Random::Marsaglia:



Public Member Functions

- unsigned long random ()

  Generate Random Number.

Static Public Member Functions

- static unsigned long * MigrateToVRAM (Markov::API::CUDA::Random::Marsaglia *MEarr, long int gridSize)

Migrate a Marsaglia[] to VRAM as seedChunk.

- static __host__ void ListCudaDevices ()

  List CUDA devices in the system.

Public Attributes

- unsigned long x
- unsigned long y
- unsigned long z

Protected Member Functions

- std::random_device & rd ()

  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.

Static Protected Member Functions

- static __host__ int CudaCheckNotifyErr (cudaError_t _status, const char *msg, bool bExit=true)

  Check results of the last operation on GPU.
- template<typename T >
  static __host__ cudaError_t CudaMalloc2DToFlat (T **dst, int row, int col)

  Malloc a 2D array in device space.
- template<typename T >
  static __host__ cudaError_t CudaMemcpy2DToFlat (T *dst, T **src, int row, int col)

  Memcpy a 2D array in device space after flattening.
- template<typename T >
  static __host__ cudaError_t CudaMigrate2DFlat (T **dst, T **src, int row, int col)

  Both malloc and memcpy a 2D array into device VRAM.

8.22.1 Detailed Description

Extension of Markov::Random::Marsaglia which is capable o working on **device** space.
Implementation of Marsaglia Random Engine. This is an implementation of Marsaglia Random engine, which for most use cases is a better fit than other solutions. Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.
This implementation of the Marsaglia Engine is seeded by random.h default random engine. RandomEngine is only seeded once so its not a performance issue.
Example Use: Using Marsaglia Engine with RandomWalk

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout « res « "\n";
 }
```

Example Use: Generating a random number with Marsaglia Engine

```
Markov::Random::Marsaglia me;
std::cout « me.random();
```

Definition at line 20 of file cudarandom.h.

8.22.2 Member Function Documentation

8.22.2.1 CudaCheckNotifyErr()

```
__host__ int Markov::API::CUDA::CUDADeviceController::CudaCheckNotifyErr (
            cudaError_t _status,
            const char * msg,
            bool bExit = true )  [static], [protected], [inherited]
```

Check results of the last operation on GPU.
Check the status returned from cudaMalloc/cudaMemcpy to find failures.
If a failure occurs, its assumed beyond redemption, and exited.

Parameters

_status	Cuda error status to check
msg	Message to print in case of a failure

Returns

0 if successful, 1 if failure. **Example output:**

```
char *da, a = "test";
cudastatus = cudaMalloc((char **)&da, 5*sizeof(char*));
CudaCheckNotifyErr(cudastatus, "Failed to allocate VRAM for *da.\n");
```

Definition at line 32 of file cudaDeviceController.cu.

```
00032
                    {
00033         if (_status != cudaSuccess) {
00034             std::cerr « "\033[1;31m" « msg  « " -> " « cudaGetErrorString(_status)  « " ("« _status «
      ")" « "\033[0m" « "\n";
00035
00036             if(bExit) {
00037                 cudaDeviceReset();
00038                 exit(1);
00039             }
00040         }
00041         return 0;
00042     }
```

8.22.2.2 CudaMalloc2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat (
            T ** dst,
            int row,
            int col )  [inline], [static], [protected], [inherited]
```

Malloc a 2D array in device space.
This function will allocate enough space on VRAM for flattened 2D array.

Parameters

dst	destination pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

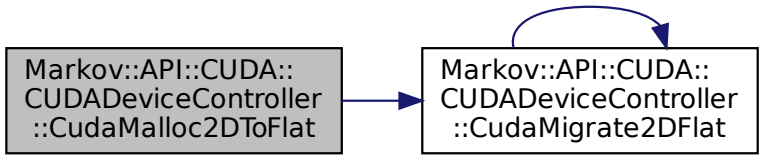
Example output:

```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess){
    CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
}
```

Definition at line 75 of file cudaDeviceController.h.

```
00075                                                                                                   {
00076             cudaError_t cudastatus = cudaMalloc((T **)dst, row*col*sizeof(T));
00077             CudaCheckNotifyErr(cudastatus, "cudaMalloc Failed.", false);
00078             return cudastatus;
00079         }
```

References Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().
Here is the call graph for this function:



8.22.2.3 CudaMemcpy2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat (
             T * dst,
             T ** src,
             int row,
             int col )  [inline], [static], [protected], [inherited]
```

Memcpy a 2D array in device space after flattening.
Resulting buffer will not be true 2D array.

Parameters

dst	destination pointer
rc	source pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

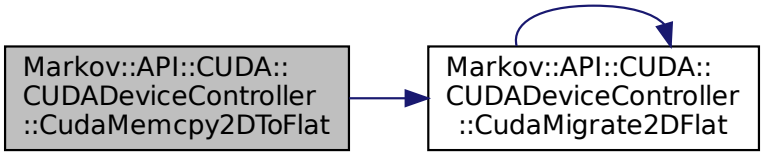
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
CudaCheckNotifyErr(cudastatus, "  CudaMemcpy2DToFlat Failed.", false);
```

Definition at line 103 of file cudaDeviceController.h.

```
00103                                                                                                   {
00104             T* tempbuf = new T[row*col];
00105             for(int i=0;i<row;i++){
00106                 memcpy(&(tempbuf[row*i]), src[i], col);
00107             }
00108             return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00109
00110         }
```

References Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().

Here is the call graph for this function:



8.22.2.4 CudaMigrate2DFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat (
            T ** dst,
            T ** src,
            int row,
            int col )  [inline], [static], [protected], [inherited]
```

Both malloc and memcpy a 2D array into device VRAM.
Resulting buffer will not be true 2D array.

Parameters

dst	destination pointer
rc	source pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

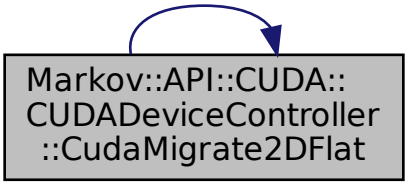
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMigrate2DFlat<long int>(
   &dst, this->valueMatrix, this->matrixSize, this->matrixSize);
CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
```

Definition at line 132 of file cudaDeviceController.h.

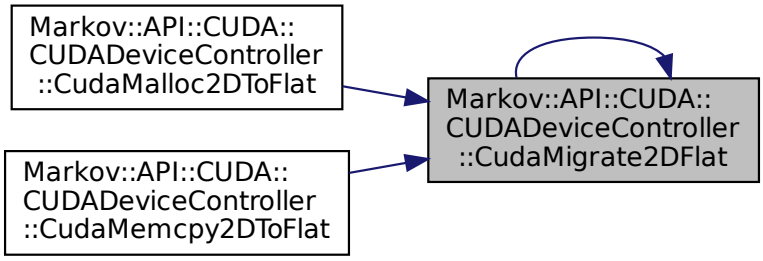
```
00132                                                                                {
00133             cudaError_t cudastatus;
00134             cudastatus = CudaMalloc2DToFlat<T>(dst, row, col);
00135             if(cudastatus!=cudaSuccess){
00136                 CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
00137                 return cudastatus;
00138             }
00139             cudastatus = CudaMemcpy2DToFlat<T>(*dst,src,row,col);
00140             CudaCheckNotifyErr(cudastatus, "  CudaMemcpy2DToFlat Failed.", false);
00141             return cudastatus;
00142         }
```

References Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().
Referenced by Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat(), Markov::API::CUDA::CUDADeviceController::Cu
and Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat().

Here is the call graph for this function:



Here is the caller graph for this function:



8.22.2.5 distribution()

`std::uniform_int_distribution<long long unsigned>& Markov::Random::DefaultRandomEngine::distribution ( ) [inline], [protected], [inherited]`

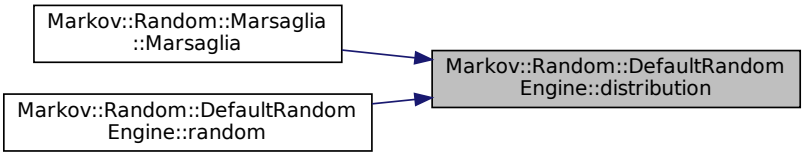
Distribution schema for seeding.

Definition at line 90 of file random.h.

```
00090                                                                   {
00091             static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffFFFF);
00092             return _distribution;
00093         }
```

Referenced by Markov::Random::Marsaglia::Marsaglia(), and Markov::Random::DefaultRandomEngine::random().

Here is the caller graph for this function:



8.22.2.6 generator()

```
std::default_random_engine& Markov::Random::DefaultRandomEngine::generator ( )  [inline],
[protected], [inherited]
```

Default random engine for seeding.

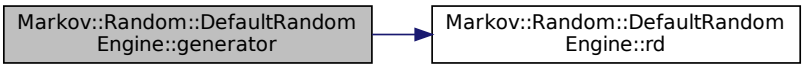
Definition at line 82 of file random.h.

```
00082                                                {
00083             static std::default_random_engine _generator(rd()());
00084             return _generator;
00085         }
```

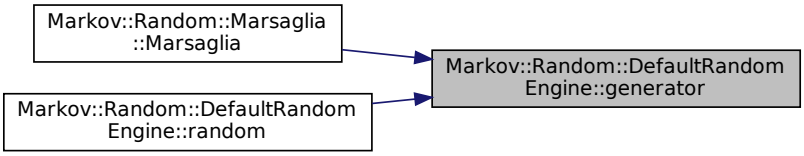
References Markov::Random::DefaultRandomEngine::rd().

Referenced by Markov::Random::Marsaglia::Marsaglia(), and Markov::Random::DefaultRandomEngine::random().

Here is the call graph for this function:



Here is the caller graph for this function:



8.22.2.7 ListCudaDevices()

```
__host__ void Markov::API::CUDA::CUDADeviceController::ListCudaDevices ( )  [static], [inherited]
```

List CUDA devices in the system.

This function will print details of every CUDA capable device in the system.

Example output:

```
Device Number: 0
Device name: GeForce RTX 2070
Memory Clock Rate (KHz): 7001000
Memory Bus Width (bits): 256
Peak Memory Bandwidth (GB/s): 448.064
Max Linear Threads: 1024
```

Definition at line 16 of file cudaDeviceController.cu.

```
00016                                                       { //list cuda Capable devices on
      host.
00017         int nDevices;
00018         cudaGetDeviceCount(&nDevices);
00019         for (int i = 0; i < nDevices; i++) {
00020             cudaDeviceProp prop;
00021             cudaGetDeviceProperties(&prop, i);
00022             std::cerr « "Device Number: " «  i  « "\n";
00023             std::cerr « "Device name: " « prop.name « "\n";
00024             std::cerr « "Memory Clock Rate (KHz): " « prop.memoryClockRate « "\n";
00025             std::cerr « "Memory Bus Width (bits): " « prop.memoryBusWidth « "\n";
00026             std::cerr « "Peak Memory Bandwidth (GB/s): " « 2.0 * prop.memoryClockRate *
      (prop.memoryBusWidth / 8) / 1.0e6 « "\n";
00027             std::cerr « "Max Linear Threads: " « prop.maxThreadsDim[0] « "\n";
00028
00029         }
00030     }
```

8.22.2.8 MigrateToVRAM()

```
static unsigned long* Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM (
            Markov::API::CUDA::Random::Marsaglia * MEarr,
            long int gridSize )  [inline], [static]
```

Migrate a Marsaglia[] to VRAM as seedChunk.

Parameters

MEarr	Array of Marsaglia Engines
gridSize	GridSize of the CUDA Kernel, aka size of array

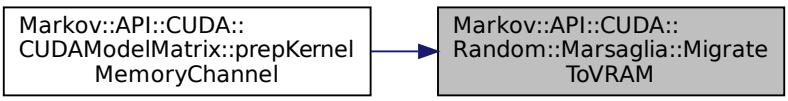
Returns

pointer to the resulting seed chunk in device VRAM.

Definition at line 28 of file cudarandom.h.

```
00028                                                                                                                {
00029             cudaError_t cudastatus;
00030             unsigned long* seedChunk;
00031             cudastatus = cudaMalloc((unsigned long**)&seedChunk, gridSize*3*sizeof(unsigned long));
00032             CudaCheckNotifyErr(cudastatus, "Failed to allocate seed buffer");
00033             unsigned long *temp = new unsigned long[gridSize*3];
00034             for(int i=0;i<gridSize;i++){
00035                 temp[i*3]   = MEarr[i].x;
00036                 temp[i*3+1] = MEarr[i].y;
00037                 temp[i*3+2] = MEarr[i].z;
00038             }
00039             //for(int i=0;i<gridSize*3;i++) std::cout << temp[i] << "\n";
00040             cudaMemcpy(seedChunk, temp, gridSize*3*sizeof(unsigned long), cudaMemcpyHostToDevice);
00041             CudaCheckNotifyErr(cudastatus, "Failed to memcpy seed buffer.");
00042
00043             delete[] temp;
00044             return seedChunk;
00045         }
```

References Markov::Random::Marsaglia::x, Markov::Random::Marsaglia::y, and Markov::Random::Marsaglia::z.
Referenced by Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel().
Here is the caller graph for this function:



8.22.2.9 random()

```
unsigned long Markov::Random::Marsaglia::random ( )  [inline], [virtual], [inherited]
```

Generate Random Number.

Returns

random number in long range.

Reimplemented from Markov::Random::DefaultRandomEngine.
Definition at line 140 of file random.h.

```
00140                                 {
00141         unsigned long t;
00142         x ^= x << 16;
00143         x ^= x >> 5;
00144         x ^= x << 1;
00145
00146         t = x;
```

```
00147         x = y;
00148         y = z;
00149         z = t ^ x ^ y;
00150 
00151         return z;
00152     }
```

References Markov::Random::Marsaglia::x, Markov::Random::Marsaglia::y, and Markov::Random::Marsaglia::z.
Referenced by Markov::API::ModelMatrix::FastRandomWalkThread().
Here is the caller graph for this function:

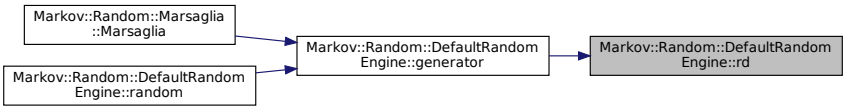
8.22.2.10 rd()

```
std::random_device& Markov::Random::DefaultRandomEngine::rd ( )  [inline], [protected], [inherited]
```

Default random device for seeding.
Definition at line 74 of file random.h.

```
00074                                   {
00075             static std::random_device _rd;
00076             return _rd;
00077         }
```

Referenced by Markov::Random::DefaultRandomEngine::generator().
Here is the caller graph for this function:



8.22.3 Member Data Documentation

8.22.3.1 x

```
unsigned long Markov::Random::Marsaglia::x  [inherited]
```

Definition at line 155 of file random.h.
Referenced by Markov::Random::Marsaglia::Marsaglia(), MigrateToVRAM(), and Markov::Random::Marsaglia::random().

8.22.3.2 y

```
unsigned long Markov::Random::Marsaglia::y  [inherited]
```

Definition at line 156 of file random.h.
Referenced by Markov::Random::Marsaglia::Marsaglia(), MigrateToVRAM(), and Markov::Random::Marsaglia::random().

8.22.3.3 z

```
unsigned long Markov::Random::Marsaglia::z  [inherited]
```

Definition at line 157 of file random.h.
Referenced by Markov::Random::Marsaglia::Marsaglia(), MigrateToVRAM(), and Markov::Random::Marsaglia::random().
The documentation for this class was generated from the following file:

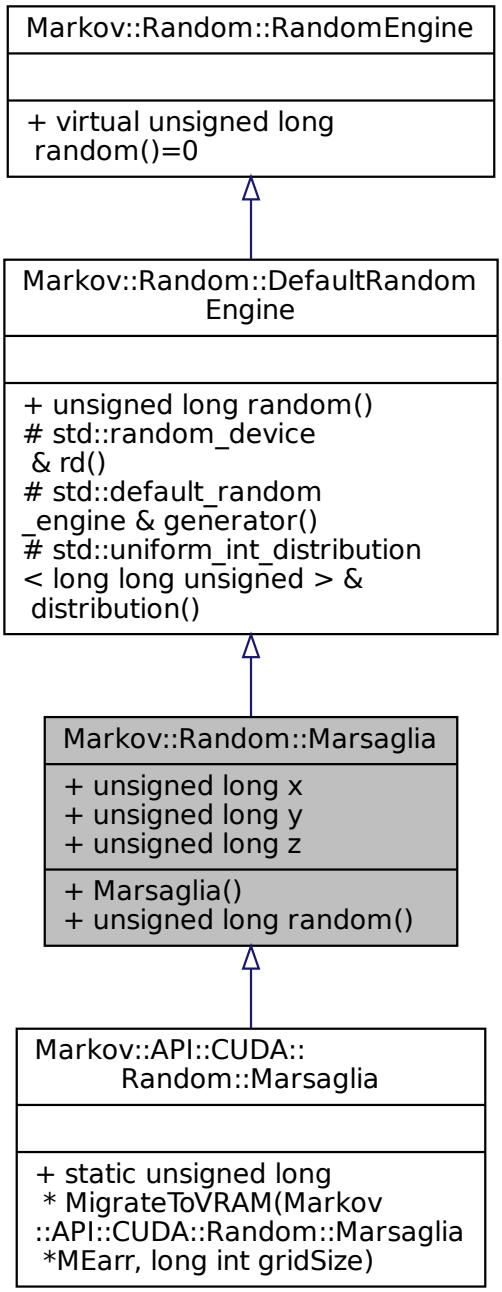
- Markopy/CudaMarkovAPI/src/cudarandom.h

8.23 Markov::Random::Marsaglia Class Reference

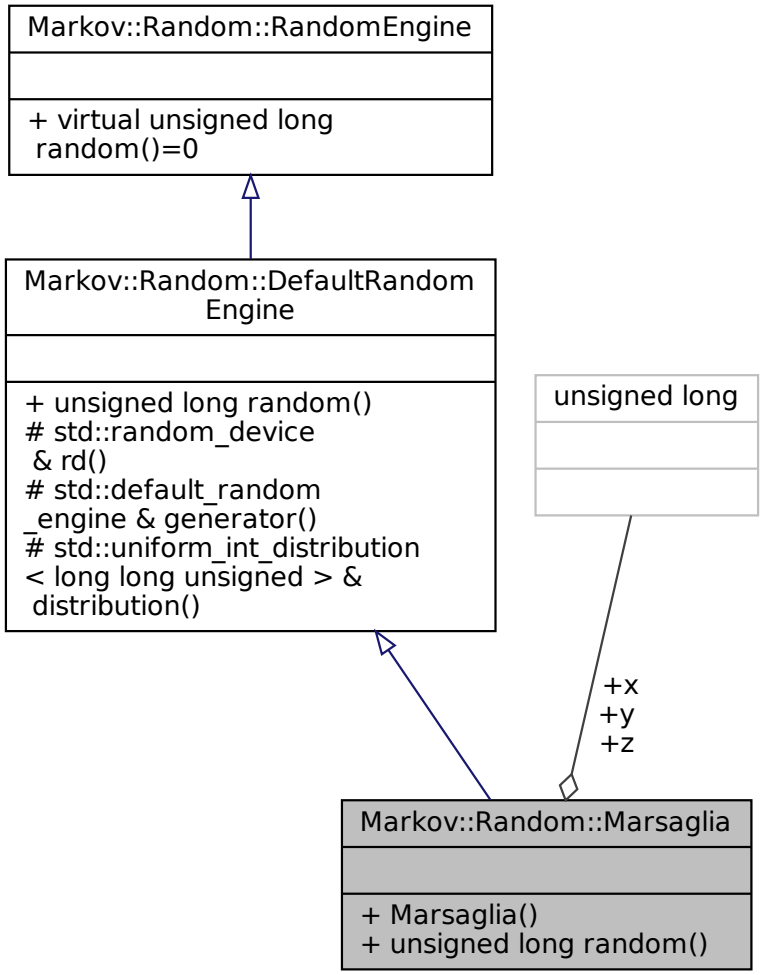
Implementation of Marsaglia Random Engine.

`#include <random.h>`

Inheritance diagram for Markov::Random::Marsaglia:



Collaboration diagram for Markov::Random::Marsaglia:



Public Member Functions

- Marsaglia ()

  Construct Marsaglia Engine.
- unsigned long random ()

  Generate Random Number.

Public Attributes

- unsigned long x
- unsigned long y
- unsigned long z

Protected Member Functions

- std::random_device & rd ()

  Default random device for seeding.

- std::default_random_engine & generator ()

  Default random engine for seeding.
- std::uniform_int_distribution< long long unsigned > & distribution ()

  Distribution schema for seeding.

8.23.1 Detailed Description

Implementation of Marsaglia Random Engine.
This is an implementation of Marsaglia Random engine, which for most use cases is a better fit than other solutions.
Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.
This implementation of the Marsaglia Engine is seeded by random.h default random engine. RandomEngine is only seeded once so its not a performance issue.
Example Use: Using Marsaglia Engine with RandomWalk

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout « res « "\n";
 }
```

Example Use: Generating a random number with Marsaglia Engine

```
Markov::Random::Marsaglia me;
std::cout « me.random();
```

Definition at line 125 of file random.h.

8.23.2 Constructor & Destructor Documentation

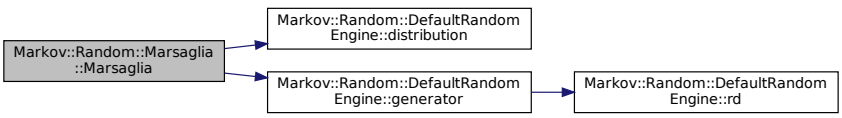
8.23.2.1 Marsaglia()

```
Markov::Random::Marsaglia::Marsaglia ( )  [inline]
```

Construct Marsaglia Engine.
Initialize x,y and z using the default random engine.
Definition at line 132 of file random.h.

```
00132                    {
00133             this->x = this->distribution()(this->generator());
00134             this->y = this->distribution()(this->generator());
00135             this->z = this->distribution()(this->generator());
00136             //std::cout « "x: " « x « ", y: " « y « ", z: " « z « "\n";
00137         }
```

References Markov::Random::DefaultRandomEngine::distribution(), Markov::Random::DefaultRandomEngine::generator(), x, y, and z.
Here is the call graph for this function:



8.23.3 Member Function Documentation

8.23.3.1 distribution()

```
std::uniform_int_distribution<long long unsigned>& Markov::Random::DefaultRandomEngine::distribution
( )  [inline], [protected], [inherited]
```

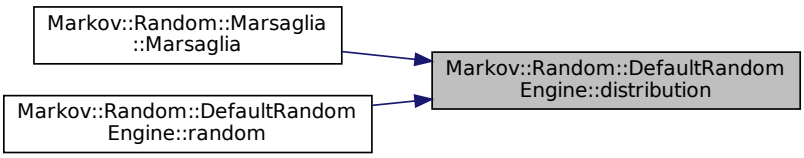
Distribution schema for seeding.

Definition at line 90 of file random.h.

```
00090                                                                   {
00091             static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffFFFF);
00092             return _distribution;
00093         }
```

Referenced by Marsaglia(), and Markov::Random::DefaultRandomEngine::random().

Here is the caller graph for this function:



8.23.3.2 generator()

`std::default_random_engine& Markov::Random::DefaultRandomEngine::generator ( )  [inline], [protected], [inherited]`

Default random engine for seeding.

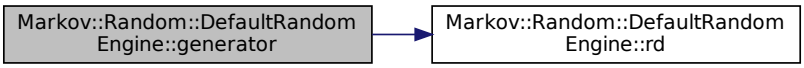
Definition at line 82 of file random.h.

```
00082                                           {
00083             static std::default_random_engine _generator(rd()());
00084             return _generator;
00085         }
```

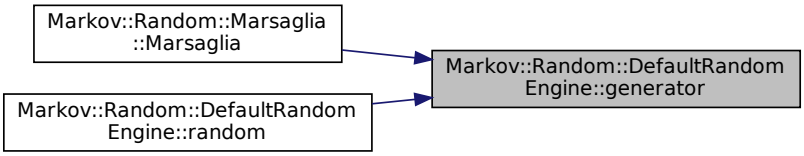
References Markov::Random::DefaultRandomEngine::rd().

Referenced by Marsaglia(), and Markov::Random::DefaultRandomEngine::random().

Here is the call graph for this function:



Here is the caller graph for this function:



8.23.3.3 random()

`unsigned long Markov::Random::Marsaglia::random ( )  [inline], [virtual]`

Generate Random Number.

Returns

random number in long range.

Reimplemented from Markov::Random::DefaultRandomEngine.
Definition at line 140 of file random.h.

```
00140                                   {
00141         unsigned long t;
00142         x ^= x « 16;
00143         x ^= x » 5;
00144         x ^= x « 1;
00145
00146         t = x;
00147         x = y;
00148         y = z;
00149         z = t ^ x ^ y;
00150
00151         return z;
00152     }
```

References x, y, and z.
Referenced by Markov::API::ModelMatrix::FastRandomWalkThread().
Here is the caller graph for this function:

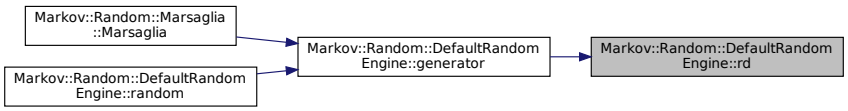
8.23.3.4 rd()

```
std::random_device& Markov::Random::DefaultRandomEngine::rd ( )  [inline], [protected], [inherited]
```

Default random device for seeding.
Definition at line 74 of file random.h.

```
00074                                   {
00075             static std::random_device _rd;
00076             return _rd;
00077         }
```

Referenced by Markov::Random::DefaultRandomEngine::generator().
Here is the caller graph for this function:



8.23.4 Member Data Documentation

8.23.4.1 x

```
unsigned long Markov::Random::Marsaglia::x
```

Definition at line 155 of file random.h.
Referenced by Marsaglia(), Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(), and random().

8.23.4.2 y

```
unsigned long Markov::Random::Marsaglia::y
```

Definition at line 156 of file random.h.
Referenced by Marsaglia(), Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(), and random().

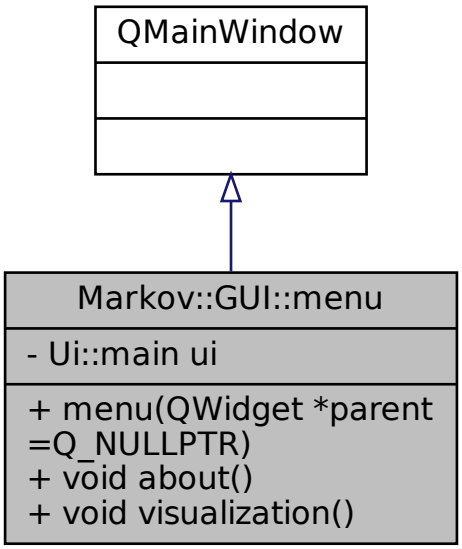
8.23.4.3 z

`unsigned long Markov::Random::Marsaglia::z`
Definition at line 157 of file random.h.
Referenced by Marsaglia(), Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(), and random().
The documentation for this class was generated from the following file:

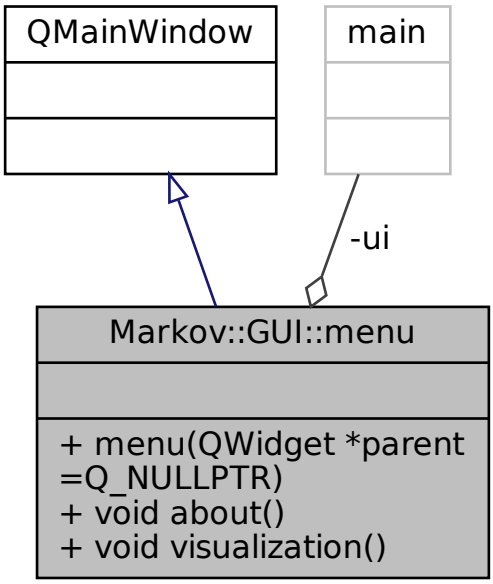
- Markopy/MarkovModel/src/random.h

8.24 Markov::GUI::menu Class Reference

QT Menu class.
`#include <menu.h>`
Inheritance diagram for Markov::GUI::menu:



Collaboration diagram for Markov::GUI::menu:



Public Slots

- void about ()
- void visualization ()

Public Member Functions

- menu (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::main ui

8.24.1 Detailed Description

QT Menu class.
Definition at line 15 of file menu.h.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 menu()

```
menu::menu (
            QWidget * parent = Q_NULLPTR )
```

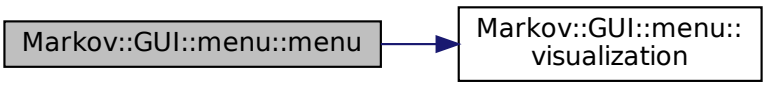
Definition at line 14 of file menu.cpp.

```
00015     : QMainWindow(parent)
00016 {
00017     ui.setupUi(this);
00018
```

```
00019
00020     //QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {about(); });
00021     QObject::connect(ui.visu, &QPushButton::clicked, this, [this] {visualization(); });
00022 }
```

References ui, and visualization().

Here is the call graph for this function:



8.24.3 Member Function Documentation

8.24.3.1 about

```
void menu::about ( )  [slot]
```

Definition at line 23 of file menu.cpp.

```
00023                 {
00024
00025
00026 }
```

8.24.3.2 visualization

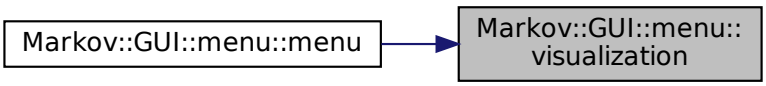
```
void menu::visualization ( )  [slot]
```

Definition at line 27 of file menu.cpp.

```
00027                         {
00028     MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
00029     w->show();
00030     this->close();
00031 }
```

Referenced by menu().

Here is the caller graph for this function:



8.24.4 Member Data Documentation

8.24.4.1 ui

```
Ui::main Markov::GUI::menu::ui  [private]
```

Definition at line 21 of file menu.h.

Referenced by menu().

The documentation for this class was generated from the following files:

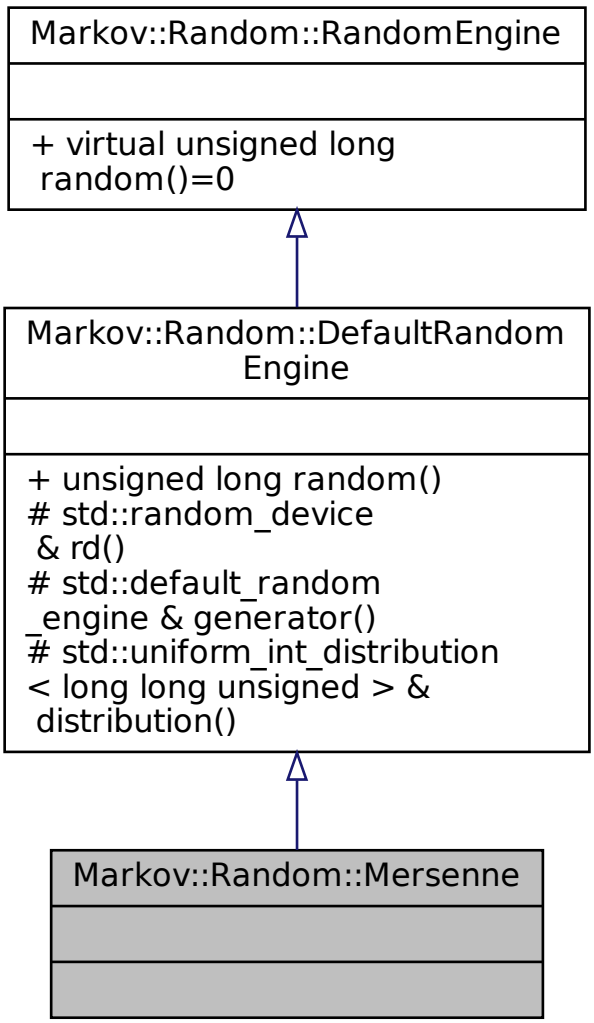
- Markopy/MarkovPasswordsGUI/src/menu.h
- Markopy/MarkovPasswordsGUI/src/menu.cpp

8.25 Markov::Random::Mersenne Class Reference

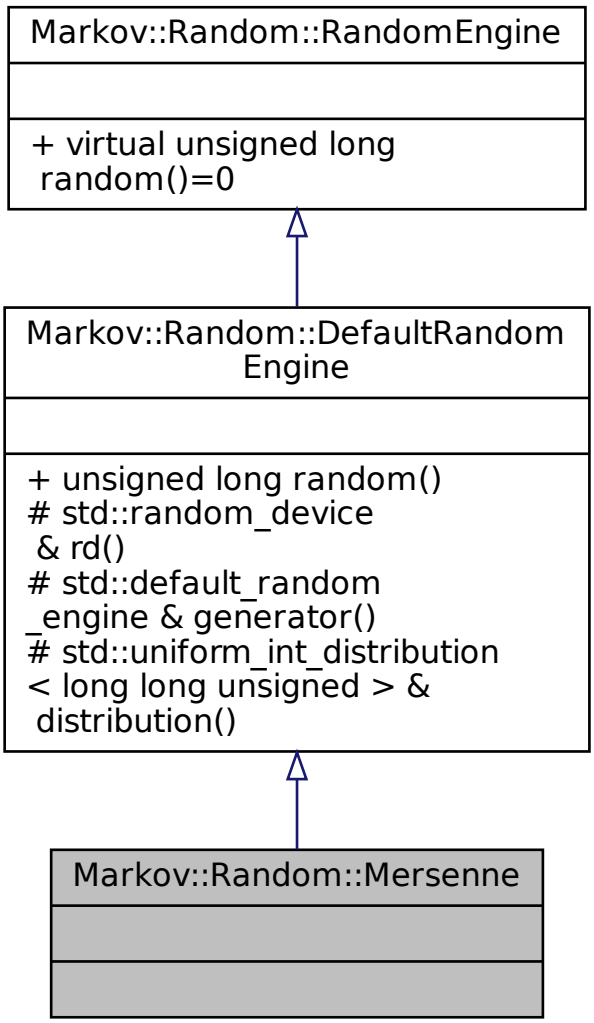
Implementation of Mersenne Twister Engine.

`#include <random.h>`

Inheritance diagram for Markov::Random::Mersenne:



Collaboration diagram for Markov::Random::Mersenne:



Public Member Functions

- unsigned long random ()

  Generate Random Number.

Protected Member Functions

- std::random_device & rd ()

  Default random device for seeding.

- std::default_random_engine & generator ()

  Default random engine for seeding.

- std::uniform_int_distribution< long long unsigned > & distribution ()

  Distribution schema for seeding.

8.25.1 Detailed Description

Implementation of Mersenne Twister Engine.
This is an implementation of Mersenne Twister Engine, which is slow but is a good implementation for high entropy pseudorandom.
Example Use: Using Mersenne Engine with RandomWalk

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Mersenne MersenneTwisterEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MersenneTwisterEngine, 5, 10, res);
    std::cout « res « "\n";
 }
```

Example Use: Generating a random number with Marsaglia Engine

```
Markov::Random::Mersenne me;
std::cout « me.random();
```

Definition at line 185 of file random.h.

8.25.2 Member Function Documentation

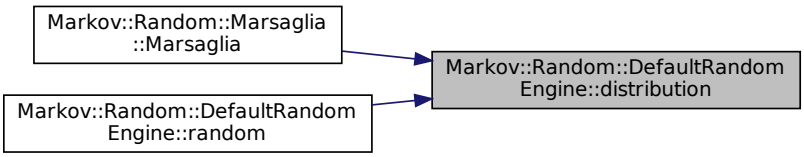
8.25.2.1 distribution()

`std::uniform_int_distribution<long long unsigned>& Markov::Random::DefaultRandomEngine::distribution ( ) [inline], [protected], [inherited]`

Distribution schema for seeding.
Definition at line 90 of file random.h.

```
00090                                                                    {
00091             static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffFFFF);
00092             return _distribution;
00093         }
```

Referenced by Markov::Random::Marsaglia::Marsaglia(), and Markov::Random::DefaultRandomEngine::random().
Here is the caller graph for this function:



8.25.2.2 generator()

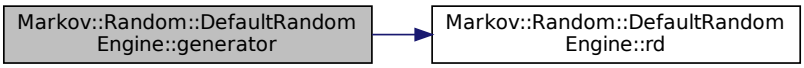
`std::default_random_engine& Markov::Random::DefaultRandomEngine::generator ( ) [inline], [protected], [inherited]`

Default random engine for seeding.
Definition at line 82 of file random.h.

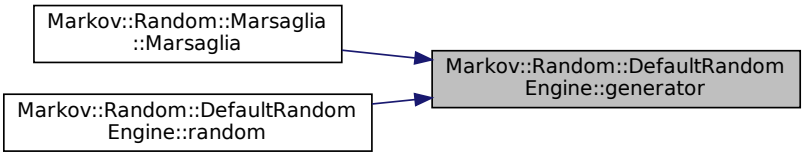
```
00082                                                {
00083             static std::default_random_engine _generator(rd()());
00084             return _generator;
00085         }
```

References Markov::Random::DefaultRandomEngine::rd().
Referenced by Markov::Random::Marsaglia::Marsaglia(), and Markov::Random::DefaultRandomEngine::random().

Here is the call graph for this function:



Here is the caller graph for this function:



8.25.2.3 random()

`unsigned long Markov::Random::DefaultRandomEngine::random ( )  [inline], [virtual], [inherited]`

Generate Random Number.

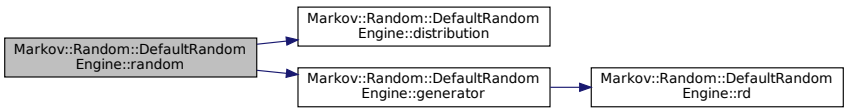
Returns

random number in long range.

Implements Markov::Random::RandomEngine.
Reimplemented in Markov::Random::Marsaglia.
Definition at line 66 of file random.h.

```
00066                                 {
00067             return this->distribution()(this->generator());
00068         }
```

References Markov::Random::DefaultRandomEngine::distribution(), and Markov::Random::DefaultRandomEngine::generator().
Here is the call graph for this function:



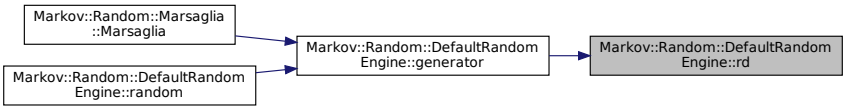
8.25.2.4 rd()

`std::random_device& Markov::Random::DefaultRandomEngine::rd ( )  [inline], [protected], [inherited]`

Default random device for seeding.
Definition at line 74 of file random.h.

```
00074                                  {
00075             static std::random_device _rd;
00076             return _rd;
00077         }
```

Referenced by Markov::Random::DefaultRandomEngine::generator().
Here is the caller graph for this function:



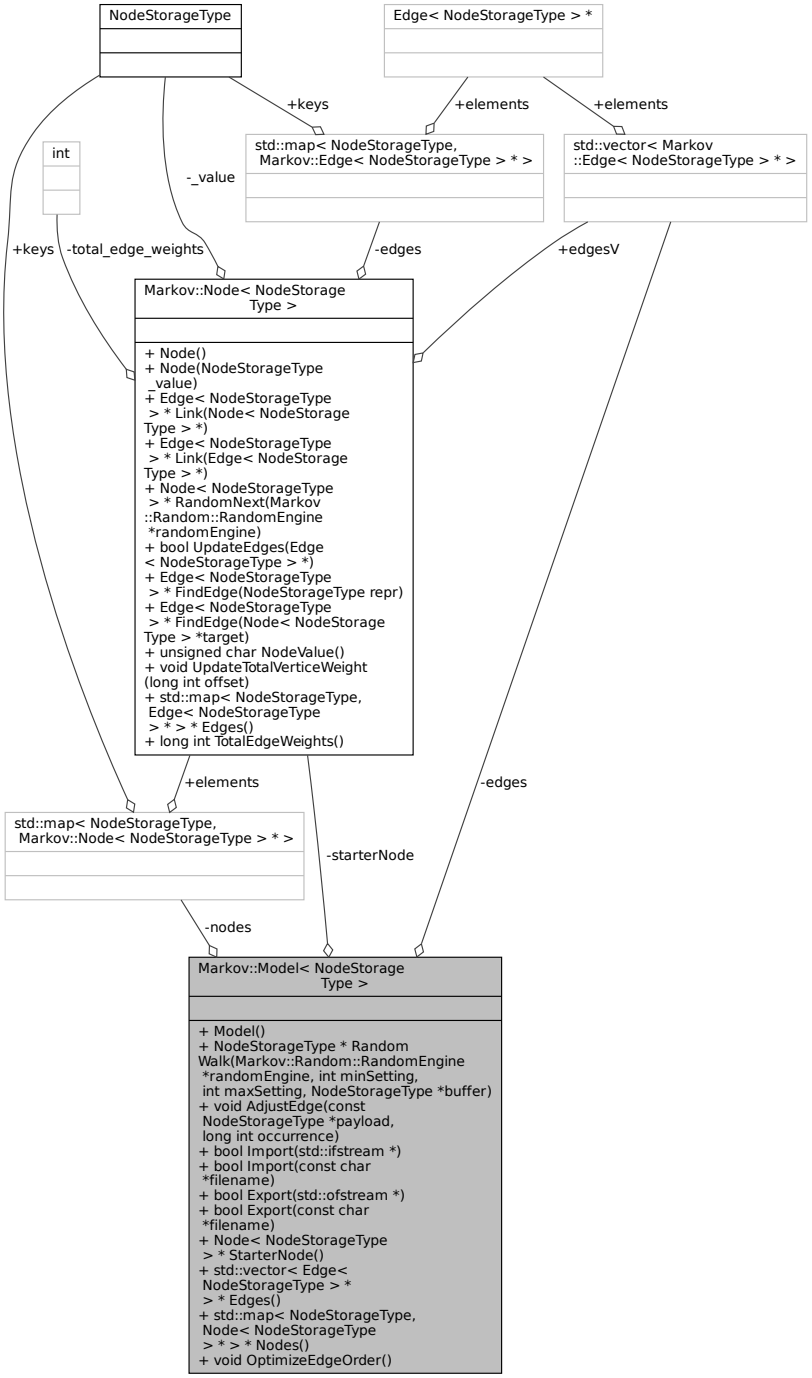
The documentation for this class was generated from the following file:

- Markopy/MarkovModel/src/random.h

8.26 Markov::Model< NodeStorageType > Class Template Reference

class for the final Markov Model, constructed from nodes and edges.
`#include <model.h>`

Collaboration diagram for Markov::Model< NodeStorageType >:



Public Member Functions

- Model ()

  Initialize a model with only start and end nodes.

- NodeStorageType * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, NodeStorageType *buffer)

  Do a random walk on this model.

- void AdjustEdge (const NodeStorageType *payload, long int occurrence)

Adjust the model with a single string.

- bool Import (std::ifstream *)

  Import a file to construct the model.
- bool Import (const char *filename)

  Open a file to import with filename, and call bool Model::Import with std::ifstream.
- bool Export (std::ofstream *)

  Export a file of the model.
- bool Export (const char *filename)

  Open a file to export with filename, and call bool Model::Export with std::ofstream.
- Node< NodeStorageType > * StarterNode ()

  Return starter Node.
- std::vector< Edge< NodeStorageType > * > * Edges ()

  Return a vector of all the edges in the model.
- std::map< NodeStorageType, Node< NodeStorageType > * > * Nodes ()

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.

Private Attributes

- std::map< NodeStorageType, Node< NodeStorageType > * > nodes

  Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< NodeStorageType > * starterNode

  Starter Node of this model.
- std::vector< Edge< NodeStorageType > * > edges

  A list of all edges in this model.

8.26.1 Detailed Description

template<typename NodeStorageType>
class Markov::Model< NodeStorageType >

class for the final Markov Model, constructed from nodes and edges.
Each atomic piece of the generation result is stored in a node, while edges contain the relation weights. *Extending:* To extend the class, implement the template and inherit from it, as "class MyModel : public Markov::Model<char>". For a complete demonstration of how to extend the class, see MarkovPasswords.
Whole model can be defined as a list of the edges, as dangling nodes are pointless. This approach is used for the import/export operations. For more information on importing/exporting model, check out the github readme and wiki page.
Definition at line 45 of file model.h.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 Model()

```
template<typename NodeStorageType >
Markov::Model< NodeStorageType >::Model
```

Initialize a model with only start and end nodes.
Initialize an empty model with only a starterNode Starter node is a special kind of node that has constant 0x00 value, and will be used to initiate the generation execution from.
Definition at line 210 of file model.h.

```
00210                                       {
00211     this->starterNode = new Markov::Node<NodeStorageType>(0);
00212     this->nodes.insert({ 0, this->starterNode });
00213 }
```

8.26.3 Member Function Documentation

8.26.3.1 AdjustEdge()

```
template<typename NodeStorageType >
void Markov::Model< NodeStorageType >::AdjustEdge (
            const NodeStorageType * payload,
            long int occurrence )
```

Adjust the model with a single string.
Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.
Then, update the edge EdgeWeight from current node, to the terminator node.
This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.
Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

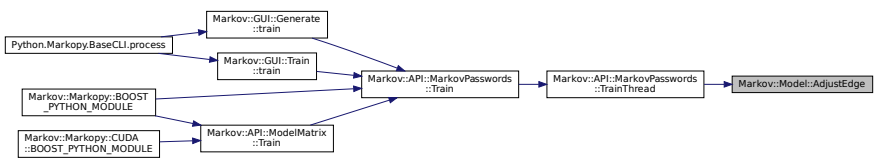
Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 337 of file model.h.

```
00337                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

Referenced by Markov::API::MarkovPasswords::TrainThread().
Here is the caller graph for this function:



8.26.3.2 Edges()

```
template<typename NodeStorageType >
std::vector<Edge<NodeStorageType>*>* Markov::Model< NodeStorageType >::Edges ( )  [inline]
```

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.26.3.3 Export() [1/2]

```
template<typename NodeStorageType >
bool Markov::Model< NodeStorageType >::Export (
            const char * filename )
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

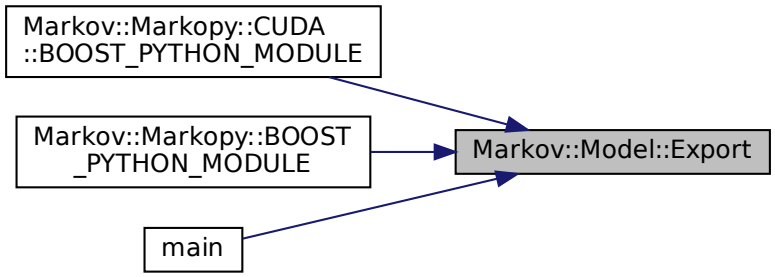
```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 300 of file model.h.

```
00300                                                          {
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(), and main().

Here is the caller graph for this function:



8.26.3.4 Export() [2/2]

```
template<typename NodeStorageType >
bool Markov::Model< NodeStorageType >::Export (
            std::ofstream * f )
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

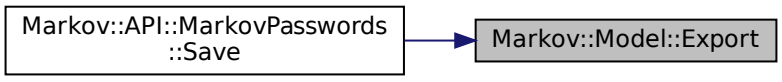
```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 288 of file model.h.

```
00288                                                   {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
      e->RightNode()->NodeValue() << "\n";
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
      "\n";
00294     }
00295
00296     return true;
00297 }
```

Referenced by Markov::API::MarkovPasswords::Save().

Here is the caller graph for this function:



8.26.3.5 Import() [1/2]

```
template<typename NodeStorageType >
bool Markov::Model< NodeStorageType >::Import (
            const char * filename )
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

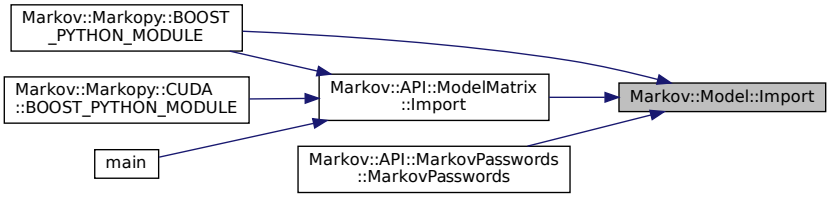
```
Markov::Model<char> model;
model.Import("test.mdl");
```

Definition at line 280 of file model.h.

```
00280                                                   {
00281     std::ifstream importfile;
00282     importfile.open(filename);
00283     return this->Import(&importfile);
00284
00285 }
```

Referenced by Markov::Markopy::BOOST_PYTHON_MODULE(), Markov::API::ModelMatrix::Import(), and Markov::API::MarkovPasswords::MarkovPasswords().

Here is the caller graph for this function:



8.26.3.6 Import() [2/2]

```
template<typename NodeStorageType >
bool Markov::Model< NodeStorageType >::Import (
            std::ifstream * f )
```

Import a file to construct the model.
File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr
Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

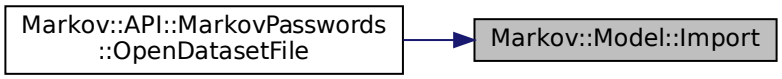
Definition at line 216 of file model.h.

```
00216                                                    {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout « "cell: " « cell « std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout « oc « "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout « "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout « "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
```

```
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

Referenced by Markov::API::MarkovPasswords::OpenDatasetFile().

Here is the caller graph for this function:



8.26.3.7 Nodes()

```
template<typename NodeStorageType >
std::map<NodeStorageType, Node<NodeStorageType>*>* Markov::Model< NodeStorageType >::Nodes (
)  [inline]
```

Return starter Node.

Returns

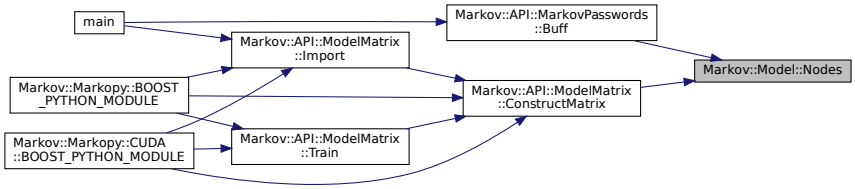
starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

Referenced by Markov::API::MarkovPasswords::Buff(), and Markov::API::ModelMatrix::ConstructMatrix().

Here is the caller graph for this function:



8.26.3.8 OptimizeEdgeOrder()

```
template<typename NodeStorageType >
void Markov::Model< NodeStorageType >::OptimizeEdgeOrder
```

Sort edges of all nodes in the model ordered by edge weights.

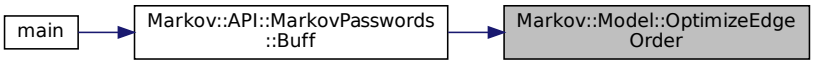
Definition at line 265 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
```

```
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

Referenced by Markov::API::MarkovPasswords::Buff().

Here is the caller graph for this function:



8.26.3.9 RandomWalk()

```
template<typename NodeStorageType >
NodeStorageType * Markov::Model< NodeStorageType >::RandomWalk (
            Markov::Random::RandomEngine * randomEngine,
            int minSetting,
            int maxSetting,
            NodeStorageType * buffer )
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length 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

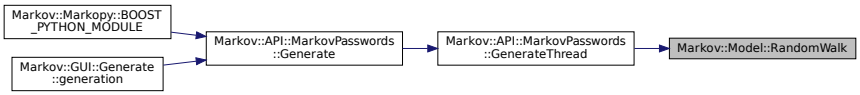
Null terminated string that was generated.

Definition at line 307 of file model.h.

```
00307                                                   {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
```

```
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319 
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323 
00324         n = temp_node;
00325 
00326         buffer[len++] = n->NodeValue();
00327     }
00328 
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331 
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

Referenced by Markov::API::MarkovPasswords::GenerateThread().
Here is the caller graph for this function:



8.26.3.10 StarterNode()

```
template<typename NodeStorageType >
Node<NodeStorageType>* Markov::Model< NodeStorageType >::StarterNode ( )  [inline]
```

Return starter Node.

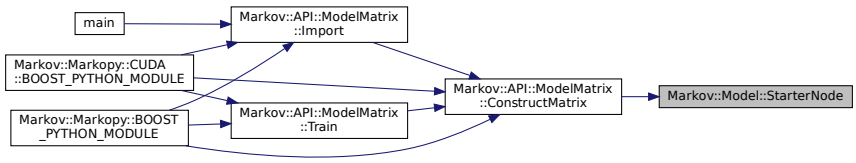
Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

Referenced by Markov::API::ModelMatrix::ConstructMatrix().
Here is the caller graph for this function:



8.26.4 Member Data Documentation

8.26.4.1 edges

```
template<typename NodeStorageType >
std::vector<Edge<NodeStorageType>*> Markov::Model< NodeStorageType >::edges  [private]
```

A list of all edges in this model.
Definition at line 204 of file model.h.
Referenced by Markov::Model< char >::Edges().

8.26.4.2 nodes

```
template<typename NodeStorageType >
std::map<NodeStorageType, Node<NodeStorageType>*> Markov::Model< NodeStorageType >::nodes
[private]
```

Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.
Referenced by Markov::Model< char >::Nodes().

8.26.4.3 starterNode

```
template<typename NodeStorageType >
Node<NodeStorageType>* Markov::Model< NodeStorageType >::starterNode  [private]
```

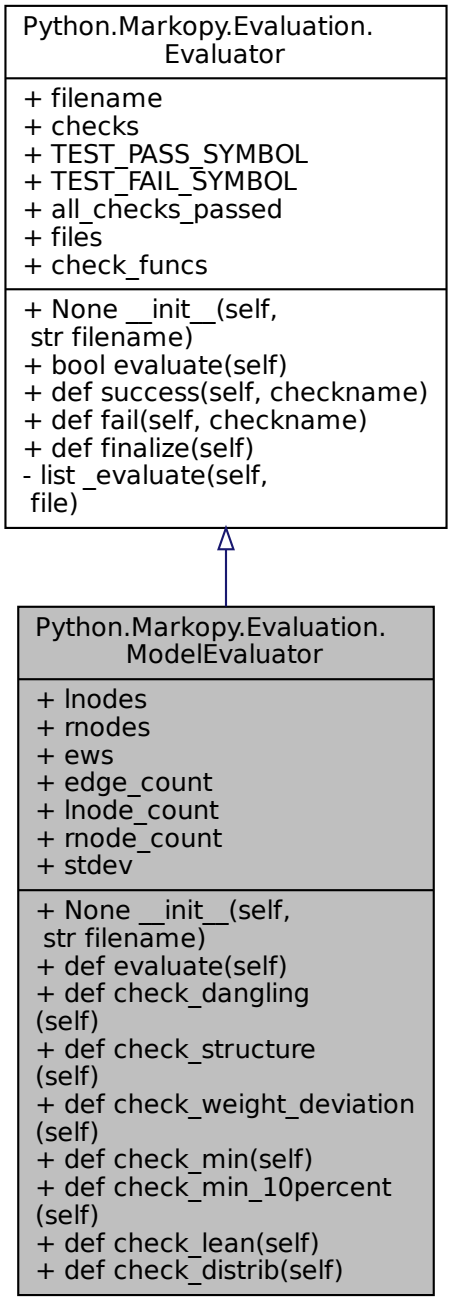
Starter Node of this model.
Definition at line 198 of file model.h.
Referenced by Markov::Model< char >::StarterNode().
The documentation for this class was generated from the following file:

- Markopy/MarkovModel/src/model.h

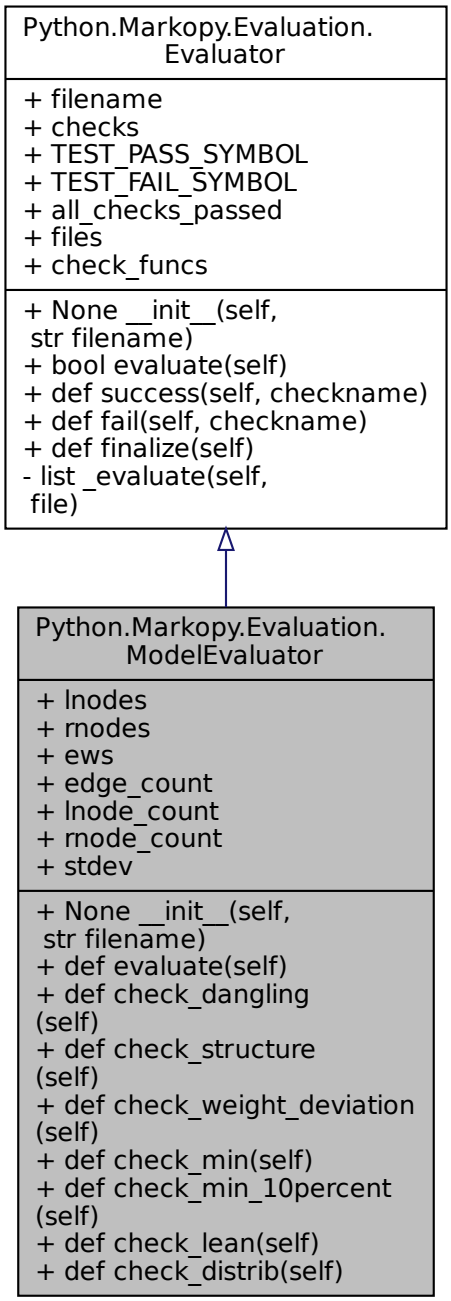
8.27 Python.Markopy.Evaluation.ModelEvaluator Class Reference

evaluate a model

Inheritance diagram for Python.Markopy.Evaluation.ModelEvaluator:



Collaboration diagram for Python.Markopy.Evaluation.ModelEvaluator:



Public Member Functions

- None __init__ (self, str filename)

  default constructor for evaluator
- def evaluate (self)
- def check_dangling (self)
- def check_structure (self)
- def check_weight_deviation (self)

- def check_min (self)
- def check_min_10percent (self)
- def check_lean (self)
- def check_distrib (self)
- def success (self, checkname)

  pass a test
- def fail (self, checkname)

  fail a test
- def finalize (self)

Public Attributes

- lnodes
- rnodes
- ews
- edge_count
- lnode_count
- rnode_count
- stdev
- filename
- checks
- TEST_PASS_SYMBOL
- TEST_FAIL_SYMBOL
- all_checks_passed
- files
- check_funcs

Private Member Functions

- list _evaluate (self, file)

  internal evaluation function for a single file

8.27.1 Detailed Description

evaluate a model
Definition at line 90 of file evaluate.py.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 __init__()

```
None Python.Markopy.Evaluation.ModelEvaluator.__init__ (
              self,
             str filename )
```

default constructor for evaluator

Parameters

filename	filename to evaluate. Can be a pattern

Reimplemented from Python.Markopy.Evaluation.Evaluator.
Definition at line 96 of file evaluate.py.

```
00096     def __init__(self, filename: str) -> None:
00097         "! @brief default constructor"
00098         valid = super().__init__(filename)
```

```
00099
00100         if not valid:
00101             return False
00102
```

8.27.3 Member Function Documentation

8.27.3.1 _evaluate()

```
list Python.Markopy.Evaluation.Evaluator._evaluate (
              self,
              file )  [private], [inherited]
```

internal evaluation function for a single file

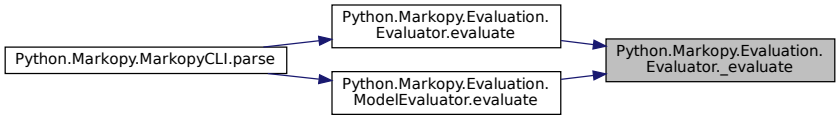
Parameters

file	filename to evaluate

Reimplemented in Python.Markopy.Evaluation.CorpusEvaluator.
Definition at line 52 of file evaluate.py.

```
00052     def _evaluate(self, file) -> list:
00053         """!
00054         @brief internal evaluation function for a single file
00055         @param file filename to evaluate
00056         """
00057         if(not os.path.isfile(file)):
00058             logging.pprint(f"Given file {file} is not a valid filename")
00059             return False
00060         else:
00061              return open(file, "rb").read().split(b"\n")
00062
00063
00064
```

Referenced by Python.Markopy.Evaluation.Evaluator.evaluate(), and Python.Markopy.Evaluation.ModelEvaluator.evaluate().
Here is the caller graph for this function:



8.27.3.2 check_dangling()

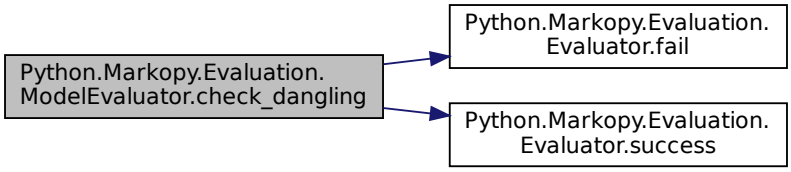
```
def Python.Markopy.Evaluation.ModelEvaluator.check_dangling (
              self )
```

Definition at line 151 of file evaluate.py.

```
00151     def check_dangling(self):
00152         "! @brief check if model has dangling nodes"
00153         if(self.lnode_count == self.rnode_count):
00154             self.success("No dangling nodes")
00155         else:
00156             logging.pprint(f"Dangling nodes found, lnodes and rnodes do not match", 0)
00157             self.fail("No dangling nodes")
00158
```

References Python.Markopy.Evaluation.Evaluator.fail(), Python.Markopy.Evaluation.ModelEvaluator.lnode_count, Python.Markopy.Evaluation.ModelEvaluator.rnode_count, and Python.Markopy.Evaluation.Evaluator.success().

Here is the call graph for this function:



8.27.3.3 check_distrib()

```
def Python.Markopy.Evaluation.ModelEvaluator.check_distrib (
                self )
```

Definition at line 228 of file evaluate.py.

```
00228     def check_distrib(self):
00229         "! @deprecated"
00230         sorted_ews = copy(self.ews)
00231         sorted_ews.sort(reverse=True)
00232         ratio1 = sorted_ews[0]/sorted_ews[int(self.edge_count/2)]
00233         ratio2 = sorted_ews[int(self.edge_count/2)]/sorted_ews[int(self.edge_count*0.1)]
00234         #print(ratio1)
00235         #print(ratio2)
00236 
00237 
```

References Python.Markopy.Evaluation.ModelEvaluator.edge_count, and Python.Markopy.Evaluation.ModelEvaluator.ews.

8.27.3.4 check_lean()

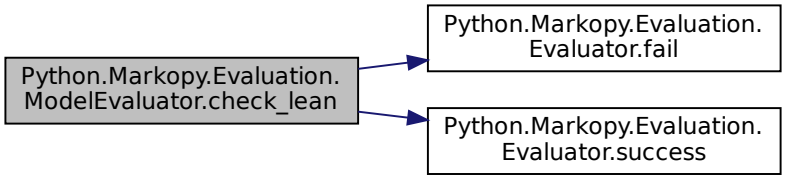
```
def Python.Markopy.Evaluation.ModelEvaluator.check_lean (
                self )
```

Definition at line 207 of file evaluate.py.

```
00207     def check_lean(self):
00208         "! @brief check which way model is leaning. Left, or right"
00209         sample = self.ews[int(self.edge_count*0.1)]
00210         avg = sum(self.ews) / len(self.ews)
00211         med = statistics.median(self.ews)
00212 
00213         if(med*10<sample):
00214             logging.pprint("Median is too left leaning and might indicate high entropy")
00215             self.fail("Median too left leaning")
00216         else:
00217             self.success("Median in expected ratio")
00218         pass
00219 
00220         if(sample*5>avg):
00221             logging.pprint("Least probable 10% too close to average, might indicate inadequate
      training")
00222             self.fail("Bad bottom 10%")
00223         else:
00224             self.success("Good bottom 10%")
00225         pass
00226 
00227 
```

References Python.Markopy.Evaluation.ModelEvaluator.edge_count, Python.Markopy.Evaluation.ModelEvaluator.ews, Python.Markopy.Evaluation.Evaluator.fail(), and Python.Markopy.Evaluation.Evaluator.success().

Here is the call graph for this function:



8.27.3.5 check_min()

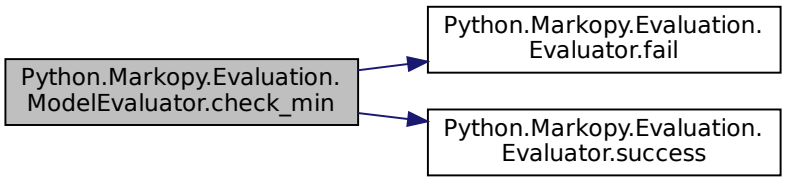
```
def Python.Markopy.Evaluation.ModelEvaluator.check_min (
              self )
```

Definition at line 186 of file evaluate.py.

```
00186     def check_min(self):
00187         "! @brief check 0 edge weights distribution"
00188         count = 0
00189         for ew in self.ews:
00190             if ew==0:
00191                 count+=1
00192         if(count > self.rnode_count*0.8):
00193             self.fail("Too many 0 edges")
00194             logging.pprint(f"0 weighted edges are dangerous and may halt the model.", 0)
00195         else:
00196             self.success("0 edges below threshold")
00197
```

References Python.Markopy.Evaluation.ModelEvaluator.ews, Python.Markopy.Evaluation.Evaluator.fail(), Python.Markopy.Evaluation.
and Python.Markopy.Evaluation.Evaluator.success().

Here is the call graph for this function:



8.27.3.6 check_min_10percent()

```
def Python.Markopy.Evaluation.ModelEvaluator.check_min_10percent (
              self )
```

Definition at line 198 of file evaluate.py.

```
00198     def check_min_10percent(self):
00199         "! @brief check minimum 10% of the edges"
00200         sample = self.ews[int(self.edge_count*0.1)]
00201         #print(f"10per: {sample}")
00202         avg = sum(self.ews) / len(self.ews)
00203         #print(f"avg: {avg}")
00204         med = statistics.median(self.ews)
00205         #print(f"med: {med}")
00206
```

References Python.Markopy.Evaluation.ModelEvaluator.edge_count, and Python.Markopy.Evaluation.ModelEvaluator.ews.

8.27.3.7 check_structure()

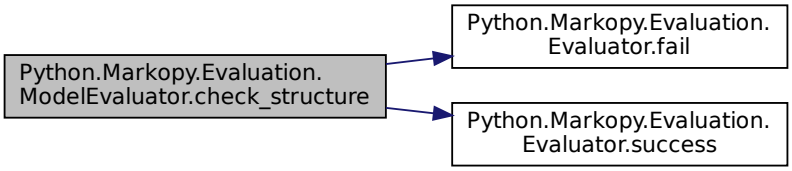
```
def Python.Markopy.Evaluation.ModelEvaluator.check_structure (
              self )
```

Definition at line 159 of file evaluate.py.

```
00159     def check_structure(self):
00160         "! @brief check model structure for validity"
00161         if((self.lnode_count-1) * (self.rnode_count-1) + 2*(self.lnode_count-1)):
00162             self.success("Model structure")
00163         else:
00164             logging.pprint(f"Model did not satisfy structural integrity check (lnode_count-1) *
      (rnode_count-1) + 2*(lnode_count-1)", 0)
00165             self.fail("Model structure")
00166
```

References Python.Markopy.Evaluation.Evaluator.fail(), Python.Markopy.Evaluation.ModelEvaluator.lnode_count, Python.Markopy.Evaluation.ModelEvaluator.rnode_count, and Python.Markopy.Evaluation.Evaluator.success().

Here is the call graph for this function:



8.27.3.8 check_weight_deviation()

```
def Python.Markopy.Evaluation.ModelEvaluator.check_weight_deviation (
              self )
```

Definition at line 167 of file evaluate.py.

```
00167     def check_weight_deviation(self):
00168         "! @brief check model standart deviation between edge weights"
00169         mean = sum(self.ews) / len(self.ews)
00170         variance = sum([(x - mean) ** 2) for x in self.ews]) / len(self.ews)
00171         res = variance ** 0.5
00172         self.stdev = res
00173         if(res==0):
00174             logging.pprint(f"Model seems to be untrained", 0)
00175             self.fail("Model has any training")
00176         else:
00177             self.success("Model has any training")
00178         if(res<3000):
00179             logging.pprint(f"Model is not adequately trained. Might result in inadequate results", 1)
00180             self.fail("Model has training")
00181             self.fail(f"Model training score: {round(self.stdev,2)}")
00182         else:
00183             self.success("Model has training")
00184             self.success(f"Model training score: {round(self.stdev)}")
00185
```

References Python.Markopy.Evaluation.ModelEvaluator.ews.

8.27.3.9 evaluate()

```
def Python.Markopy.Evaluation.ModelEvaluator.evaluate (
              self )
```

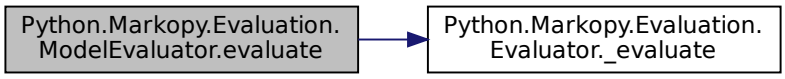
Reimplemented from Python.Markopy.Evaluation.Evaluator.

Definition at line 103 of file evaluate.py.

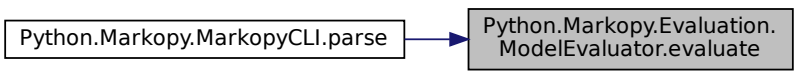
```
00103     def evaluate(self):
```

```
00104         "! @brief evaluate a model"
00105         logging.VERBOSITY=2
00106         logging.SHOW_STACK_THRESHOLD=3
00107         super().evaluate()
00108         for file in self.files:
00109             logging.pprint(f"Model: {file.split('/')[-1]}: ",2)
00110             edges = super()._evaluate(file)
00111             if not edges:
00112                 continue
00113             self.lnodes = {}
00114             self.rnodes = {}
00115             self.ews = []
00116             self.edge_count = len(edges)
00117             for edge in edges:
00118                 if(edge ==b''):
00119                     self.edge_count-=1
00120                     continue
00121                 try:
00122                     e = edge.split(b',')
00123                     self.ews.append(int(edge[2:-2:1]))
00124                     if(e[0] not in self.lnodes):
00125                         self.lnodes[e[0]]=1
00126                     else:
00127                         self.lnodes[e[0]]+=1
00128                     if(e[-1] not in self.rnodes):
00129                         self.rnodes[e[-1]]=1
00130                     else:
00131                         self.rnodes[e[-1]]+=1
00132                 except Exception as e:
00133                     print(e)
00134                     logging.pprint(f"Model file is corrupted.", 0)
00135                     continue
00136 
00137             self.lnode_count = len(self.lnodes)
00138             self.rnode_count = len(self.rnodes)
00139             logging.pprint(f"total edges: {self.edge_count}", 1)
00140             logging.pprint(f"unique left nodes: {self.lnode_count}", 1)
00141             logging.pprint(f"unique right nodes: {self.rnode_count}", 1)
00142 
00143             for check in self.check_funcs:
00144                 try:
00145                     self.__getattribute__(check)()
00146                 except Exception as e:
00147                     print(e)
00148                     self.fail(f"Exceptionn in {check}")
00149             self.finalize()
00150 
```

References Python.Markopy.Evaluation.Evaluator._evaluate(), and Python.Markopy.Evaluation.Evaluator.files.
Referenced by Python.Markopy.MarkopyCLI.parse().
Here is the call graph for this function:



Here is the caller graph for this function:



8.27.3.10 fail()

```
def Python.Markopy.Evaluation.Evaluator.fail (
            self,
            checkname )  [inherited]
```

fail a test

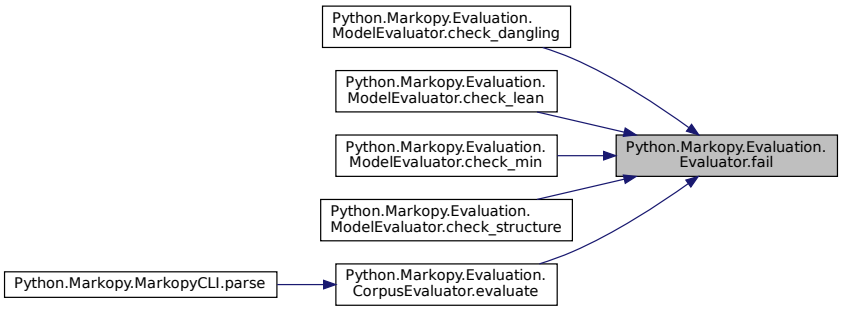
Parameters

checkname	text to display with the check

Definition at line 72 of file evaluate.py.

```
00072     def fail(self, checkname):
00073         """!
00074         @brief fail a test
00075         @param checkname text to display with the check
00076         """
00077 
00078         self.all_checks_passed = False
00079         self.checks.append((checkname, self.TEST_FAIL_SYMBOL))
00080 
```

References Python.Markopy.Evaluation.Evaluator.all_checks_passed, Python.Markopy.Evaluation.Evaluator.checks, and Python.Markopy.Evaluation.Evaluator.TEST_FAIL_SYMBOL.
Referenced by Python.Markopy.Evaluation.ModelEvaluator.check_dangling(), Python.Markopy.Evaluation.ModelEvaluator.check_lean Python.Markopy.Evaluation.ModelEvaluator.check_min(), Python.Markopy.Evaluation.ModelEvaluator.check_structure(), and Python.Markopy.Evaluation.CorpusEvaluator.evaluate().
Here is the caller graph for this function:



8.27.3.11 finalize()

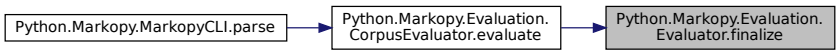
```
def Python.Markopy.Evaluation.Evaluator.finalize (
            self )  [inherited]
```

Definition at line 81 of file evaluate.py.

```
00081     def finalize(self):
00082         "! @brief finalize an evaluation and print checks"
00083         print("\n################# Checks ################ ")
00084         for test in self.checks:
00085             logging.pprint(f"{test[0]:30}:{test[1]} ")
00086         print("\n")
00087         self.checks = []
00088         return self.all_checks_passed
00089 
```

References Python.Markopy.Evaluation.Evaluator.all_checks_passed, and Python.Markopy.Evaluation.Evaluator.checks.
Referenced by Python.Markopy.Evaluation.CorpusEvaluator.evaluate().

Here is the caller graph for this function:



8.27.3.12 success()

```
def Python.Markopy.Evaluation.Evaluator.success (
              self,
              checkname )  [inherited]
```

pass a test

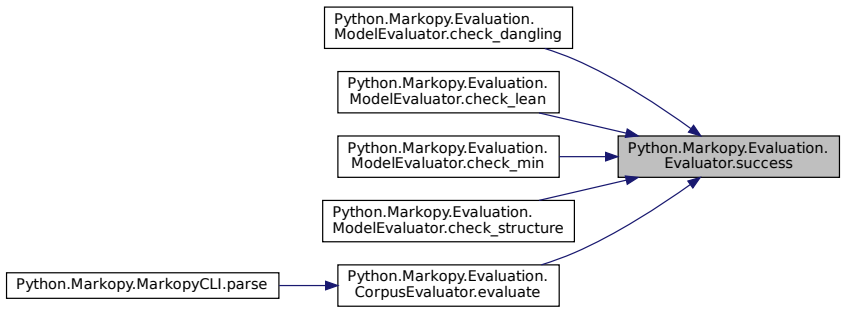
Parameters

checkname	text to display with the check

Definition at line 65 of file evaluate.py.

```
00065     def success(self, checkname):
00066         """!
00067         @brief pass a test
00068         @param checkname text to display with the check
00069         """
00070         self.checks.append((checkname, self.TEST_PASS_SYMBOL))
00071
```

References Python.Markopy.Evaluation.Evaluator.checks, and Python.Markopy.Evaluation.Evaluator.TEST_PASS_SYMBOL.
Referenced by Python.Markopy.Evaluation.ModelEvaluator.check_dangling(), Python.Markopy.Evaluation.ModelEvaluator.check_lean
Python.Markopy.Evaluation.ModelEvaluator.check_min(), Python.Markopy.Evaluation.ModelEvaluator.check_structure(),
and Python.Markopy.Evaluation.CorpusEvaluator.evaluate().
Here is the caller graph for this function:



8.27.4 Member Data Documentation

8.27.4.1 all_checks_passed

```
Python.Markopy.Evaluation.Evaluator.all_checks_passed  [inherited]
```

Definition at line 36 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.fail(), and Python.Markopy.Evaluation.Evaluator.finalize().

8.27.4.2 check_funcs

`Python.Markopy.Evaluation.Evaluator.check_funcs [inherited]`
Definition at line 49 of file evaluate.py.

8.27.4.3 checks

`Python.Markopy.Evaluation.Evaluator.checks [inherited]`
Definition at line 33 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.fail(), Python.Markopy.Evaluation.Evaluator.finalize(), and Python.Markopy.Evaluation.Evaluator.success().

8.27.4.4 edge_count

`Python.Markopy.Evaluation.ModelEvaluator.edge_count`
Definition at line 116 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.ModelEvaluator.check_distrib(), Python.Markopy.Evaluation.ModelEvaluator.check_lean(), and Python.Markopy.Evaluation.ModelEvaluator.check_min_10percent().

8.27.4.5 ews

`Python.Markopy.Evaluation.ModelEvaluator.ews`
Definition at line 115 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.ModelEvaluator.check_distrib(), Python.Markopy.Evaluation.ModelEvaluator.check_lean(), Python.Markopy.Evaluation.ModelEvaluator.check_min(), Python.Markopy.Evaluation.ModelEvaluator.check_min_10percent(), and Python.Markopy.Evaluation.ModelEvaluator.check_weight_deviation().

8.27.4.6 filename

`Python.Markopy.Evaluation.Evaluator.filename [inherited]`
Definition at line 32 of file evaluate.py.

8.27.4.7 files

`Python.Markopy.Evaluation.Evaluator.files [inherited]`
Definition at line 37 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.evaluate(), Python.Markopy.Evaluation.ModelEvaluator.evaluate(), and Python.Markopy.Evaluation.CorpusEvaluator.evaluate().

8.27.4.8 lnode_count

`Python.Markopy.Evaluation.ModelEvaluator.lnode_count`
Definition at line 137 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.ModelEvaluator.check_dangling(), and Python.Markopy.Evaluation.ModelEvaluator.check_

8.27.4.9 lnodes

`Python.Markopy.Evaluation.ModelEvaluator.lnodes`
Definition at line 113 of file evaluate.py.

8.27.4.10 rnode_count

`Python.Markopy.Evaluation.ModelEvaluator.rnode_count`
Definition at line 138 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.ModelEvaluator.check_dangling(), Python.Markopy.Evaluation.ModelEvaluator.check_min
and Python.Markopy.Evaluation.ModelEvaluator.check_structure().

8.27.4.11 rnodes

`Python.Markopy.Evaluation.ModelEvaluator.rnodes`
Definition at line 114 of file evaluate.py.

8.27.4.12 stdev

`Python.Markopy.Evaluation.ModelEvaluator.stdev`
Definition at line 172 of file evaluate.py.

8.27.4.13 TEST_FAIL_SYMBOL

`Python.Markopy.Evaluation.Evaluator.TEST_FAIL_SYMBOL [inherited]`
Definition at line 35 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.fail().

8.27.4.14 TEST_PASS_SYMBOL

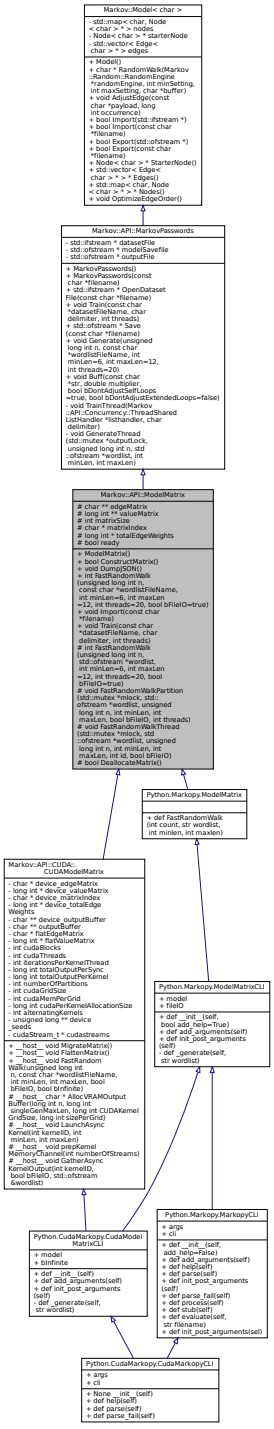
`Python.Markopy.Evaluation.Evaluator.TEST_PASS_SYMBOL [inherited]`
Definition at line 34 of file evaluate.py.
Referenced by Python.Markopy.Evaluation.Evaluator.success().
The documentation for this class was generated from the following file:

- Markopy/Markopy/src/CLI/evaluate.py

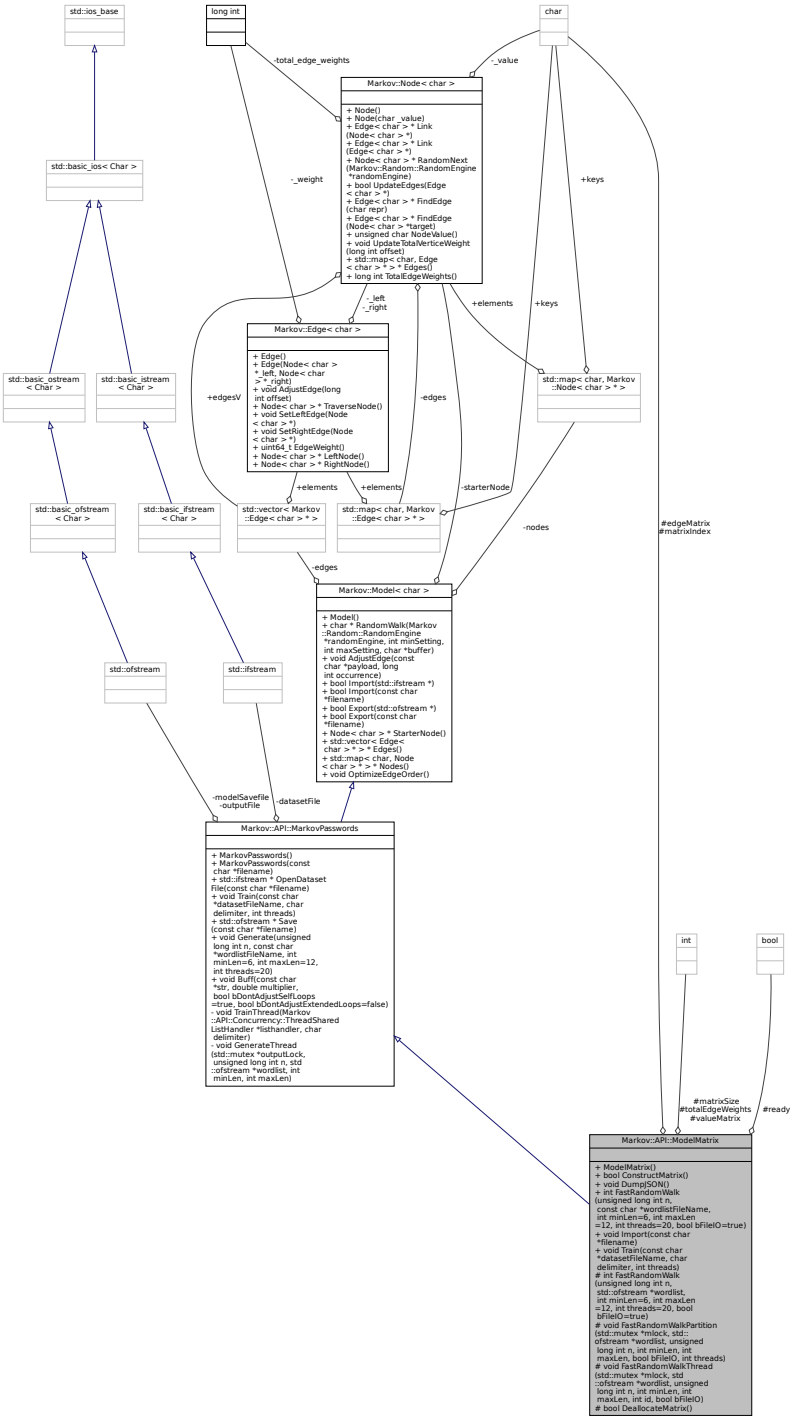
8.28 Markov::API::ModelMatrix Class Reference

Class to flatten and reduce Markov::Model to a Matrix.
`#include <modelMatrix.h>`

Inheritance diagram for Markov::API::ModelMatrix:



Collaboration diagram for Markov::API::ModelMatrix:



Public Member Functions

- ModelMatrix ()
- bool ConstructMatrix ()

  Construct the related Matrix data for the model.
- void DumpJSON ()

  Debug function to dump the model to a JSON file.

- int FastRandomWalk (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- void Import (const char *filename)

  Open a file to import with filename, and call bool Model::Import with std::ifstream.
- void Train (const char *datasetFileName, char delimiter, int threads)

  Train the model with the dataset file.
- std::ifstream * OpenDatasetFile (const char *filename)

  Open dataset file and return the ifstream pointer.
- std::ofstream * Save (const char *filename)

  Export model to file.
- void Generate (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

  Call Markov::Model::RandomWalk n times, and collect output.
- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- char * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

  Do a random walk on this model.
- void AdjustEdge (const char *payload, long int occurrence)

  Adjust the model with a single string.
- bool Import (std::ifstream *)

  Import a file to construct the model.
- bool Export (std::ofstream *)

  Export a file of the model.
- bool Export (const char *filename)

  Open a file to export with filename, and call bool Model::Export with std::ofstream.
- Node< char > * StarterNode ()

  Return starter Node.
- std::vector< Edge< char > * > * Edges ()

  Return a vector of all the edges in the model.
- std::map< char, Node< char > * > * Nodes ()

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.

Protected Member Functions

- int FastRandomWalk (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- void FastRandomWalkPartition (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

  A single partition of FastRandomWalk event.
- void FastRandomWalkThread (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)

  A single thread of a single partition of FastRandomWalk.
- bool DeallocateMatrix ()

  Deallocate matrix and make it ready for re-construction.

Protected Attributes

- char ** edgeMatrix

  2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** valueMatrix

  2-d Integer array for the value Matrix (For the weights of Edges)
- int matrixSize

  to hold Matrix size
- char * matrixIndex

  to hold the Matrix index (To hold the orders of 2-D arrays')
- long int * totalEdgeWeights

  Array of the Total Edge Weights.
- bool ready

  True when matrix is constructed. False if not.

Private Member Functions

- void TrainThread (Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char delimiter)

  A single thread invoked by the Train function.
- void GenerateThread (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)

  A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * datasetFile
- std::ofstream * modelSavefile

  Dataset file input of our system
- std::ofstream * outputFile

  File to save model of our system
- std::map< char, Node< char > * > nodes

  Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > * starterNode

  Starter Node of this model.
- std::vector< Edge< char > * > edges

  A list of all edges in this model.

8.28.1 Detailed Description

Class to flatten and reduce Markov::Model to a Matrix.
Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)
To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are sychronized and files are flushed every 50M operations.
Definition at line 23 of file modelMatrix.h.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 ModelMatrix()

```
Markov::API::ModelMatrix::ModelMatrix ( )
```

Definition at line 15 of file modelMatrix.cpp.

```
00015                                   {
00016     this->ready = false;
00017 }
```

References ready.

8.28.3 Member Function Documentation

8.28.3.1 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
            const NodeStorageType * payload,
            long int occurrence )  [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file model.h.

```
00337                                                                                                   {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.28.3.2 Buff()

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed

Parameters

multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

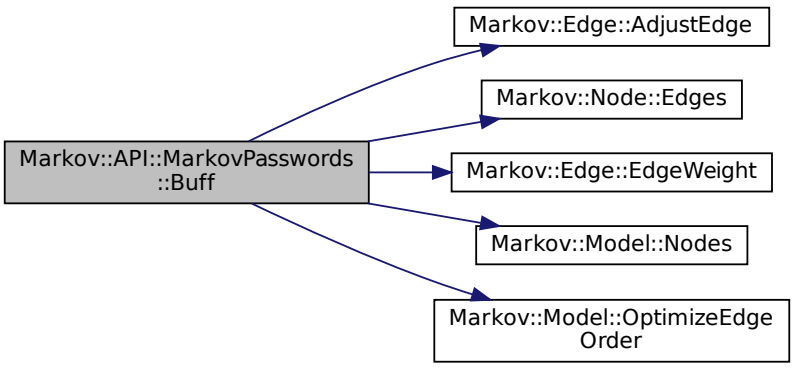
Definition at line 153 of file markovPasswords.cpp.

```
00153                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173 
00174         }
00175         i++;
00176     }
00177 
00178     this->OptimizeEdgeOrder();
00179 }
```

References Markov::Edge< NodeStorageType >::AdjustEdge(), Markov::Node< storageType >::Edges(), Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::Model< NodeStorageType >::Nodes(), and Markov::Model< NodeStorageType >::OptimizeEdgeOrder().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.28.3.3 ConstructMatrix()

```
bool Markov::API::ModelMatrix::ConstructMatrix ( )
```

Construct the related Matrix data for the model.
This operation can be used after importing/training to allocate and populate the matrix content.
this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

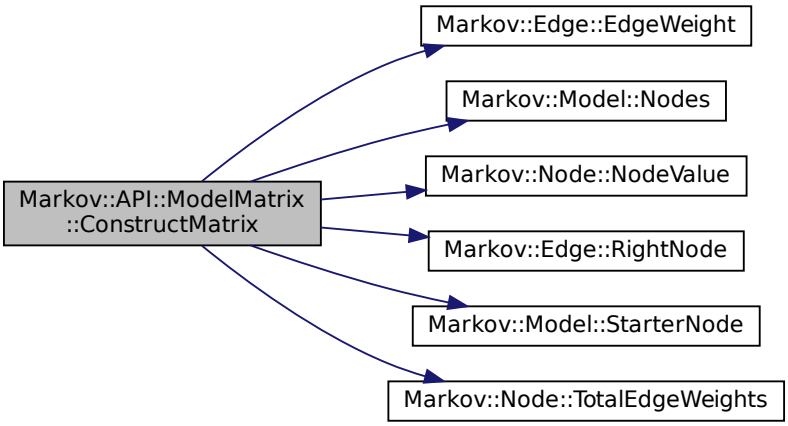
True if constructed. False if already construced.

Definition at line 31 of file modelMatrix.cpp.

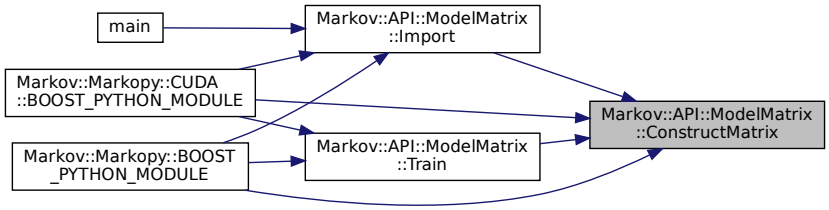
```
00031                                           {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034 
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037 
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072 
00073         }
00074         i++;
00075     }
```

```
00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

References edgeMatrix, Markov::Edge< NodeStorageType >::EdgeWeight(), matrixIndex, matrixSize, Markov::Model< NodeStorage
Markov::Node< storageType >::NodeValue(), ready, Markov::Edge< NodeStorageType >::RightNode(), Markov::Model< NodeStora
totalEdgeWeights, Markov::Node< storageType >::TotalEdgeWeights(), and valueMatrix.
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(),
Import(), and Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.28.3.4 DeallocateMatrix()

```
bool Markov::API::ModelMatrix::DeallocateMatrix ( )  [protected]
```

Deallocate matrix and make it ready for re-construction.

Returns

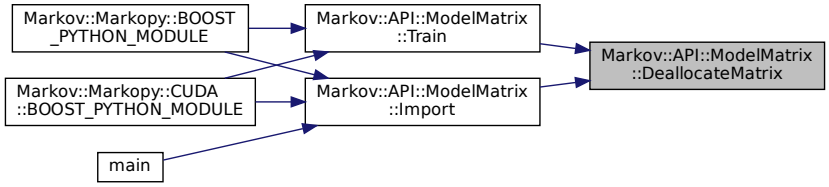
True if deallocated. False if matrix was not initialized

Definition at line 81 of file modelMatrix.cpp.

```
00081                                           {
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
00085
00086     for(int i=0;i<this->matrixSize;i++){
```

```
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090
00091     for(int i=0;i<this->matrixSize;i++){
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
00099 }
```

References edgeMatrix, matrixIndex, matrixSize, ready, totalEdgeWeights, and valueMatrix.
Referenced by Import(), and Train().
Here is the caller graph for this function:



8.28.3.5 DumpJSON()

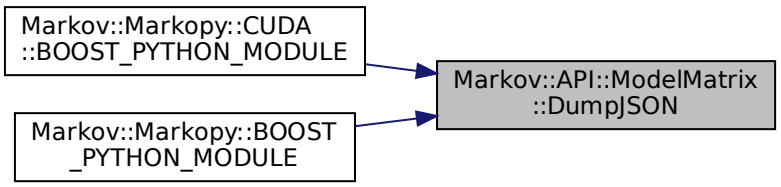
`void Markov::API::ModelMatrix::DumpJSON ( )`

Debug function to dump the model to a JSON file.
Might not work 100%. Not meant for production use.
Definition at line 101 of file modelMatrix.cpp.

```
00101                                   {
00102
00103     std::cout << "{\n   \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout << "\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout << "\\\\";
00107         else if(this->matrixIndex[i]==0) std::cout << "\\\\x00";
00108         else if(i==0) std::cout << "\\\\xff";
00109         else if(this->matrixIndex[i]=='\n') std::cout << "\\n";
00110         else std::cout << this->matrixIndex[i];
00111     }
00112     std::cout <<
00113     "\",\n"
00114     "   \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++){
00117         if(this->matrixIndex[i]=='"') std::cout << "      \"\\\"\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\x00\": [";
00120         else if(this->matrixIndex[i]<0) std::cout << "      \"\\\\xff\": [";
00121         else std::cout << "      \"" << this->matrixIndex[i] << "\": [";
00122         for(int j=0;j<this->matrixSize;j++){
00123             if(this->edgeMatrix[i][j]=='"') std::cout << "\"\\\"\"";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout << "\"\\\\\"";
00125             else if(this->edgeMatrix[i][j]==0) std::cout << "\"\\\\x00\"";
00126             else if(this->edgeMatrix[i][j]<0) std::cout << "\"\\\\xff\"";
00127             else if(this->matrixIndex[i]=='\n') std::cout << "\"\\n\"";
00128             else std::cout << "\"" << this->edgeMatrix[i][j] << "\"";
00129             if(j!=this->matrixSize-1) std::cout << ", ";
00130         }
00131         std::cout << "],\n";
00132     }
00133     std::cout << "},\n";
00134
00135     std::cout << "\"   weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++){
00137         if(this->matrixIndex[i]=='"') std::cout << "      \"\\\"\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\x00\": [";
00140         else if(this->matrixIndex[i]<0) std::cout << "      \"\\\\xff\": [";
```

```
00141         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++){
00144             std::cout « this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout « ", ";
00146         }
00147         std::cout « "],\n";
00148     }
00149     std::cout « "  }\n}\n";
00150 }
```

References edgeMatrix, matrixIndex, matrixSize, and valueMatrix.
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), and Markov::Markopy::BOOST_PYTHON_MODULE().
Here is the caller graph for this function:



8.28.3.6 Edges()

```
std::vector<Edge<char >*>* Markov::Model< char >::Edges ( )  [inline], [inherited]
```

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.28.3.7 Export() [1/2]

```
bool Markov::Model< char >::Export (
            const char * filename )  [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file model.h.

```
00300                                                   {
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

8.28.3.8 Export() [2/2]

```
bool Markov::Model< char >::Export (
            std::ofstream * f )  [inherited]
```

Export a file of the model.
File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.
Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file model.h.

```
00288                                                       {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
      e->RightNode()->NodeValue() « "\n";
00293         *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.28.3.9 FastRandomWalk() [1/2]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )
```

Random walk on the Matrix-reduced Markov::Model.
This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.
While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

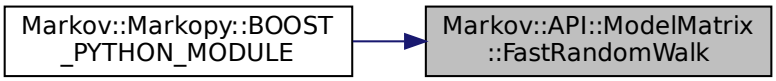
```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 217 of file modelMatrix.cpp.

```
00217
                                                {
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

References FastRandomWalk().
Referenced by Markov::Markopy::BOOST_PYTHON_MODULE().
Here is the call graph for this function:

Here is the caller graph for this function:



8.28.3.10 FastRandomWalk() [2/2]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [protected]
```

Random walk on the Matrix-reduced Markov::Model.
This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.
While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5
This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

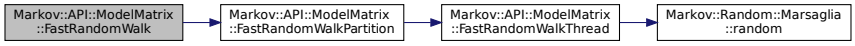
```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 204 of file modelMatrix.cpp.

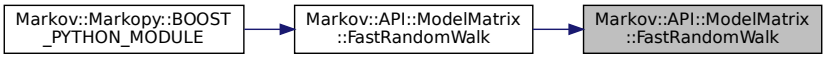
```
00204
                                    {
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
      threads);
```

```
00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
      threads);
00213     }
00214     return 0;
00215 }
```

References FastRandomWalkPartition().
Referenced by FastRandomWalk().
Here is the call graph for this function:



Here is the caller graph for this function:



8.28.3.11 FastRandomWalkPartition()

```
void Markov::API::ModelMatrix::FastRandomWalkPartition (
            std::mutex * mlock,
            std::ofstream * wordlist,
            unsigned long int n,
            int minLen,
            int maxLen,
            bool bFileIO,
            int threads )  [protected]
```

A single partition of FastRandomWalk event.
Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation
- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

Parameters

mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

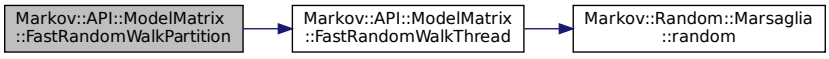
Definition at line 225 of file modelMatrix.cpp.

```
00225                                                                    {
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++){
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
      mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
      wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00239
00240     for(int i=0;i<threads;i++){
00241         threadsV[i]->join();
00242     }
00243 }
```

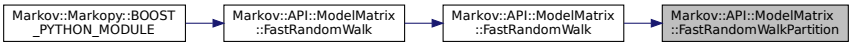
References FastRandomWalkThread().

Referenced by FastRandomWalk().

Here is the call graph for this function:



Here is the caller graph for this function:



8.28.3.12 FastRandomWalkThread()

```
void Markov::API::ModelMatrix::FastRandomWalkThread (
            std::mutex * mlock,
            std::ofstream * wordlist,
            unsigned long int n,
            int minLen,
            int maxLen,
            int id,
            bool bFileIO )  [protected]
```

A single thread of a single partition of FastRandomWalk.

A FastRandomWalkPartition will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

Parameters

id	- **DEPRECATED** Thread id - No longer used
bFileIO	- If false, filename will be ignored and will output to stdout.

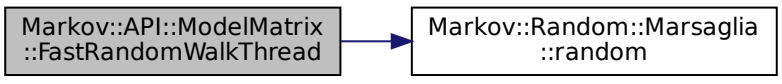
Definition at line 153 of file modelMatrix.cpp.

```
00153                                                       {
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179
00180             if (len >= maxLen)  break;
00181             else if ((next < 0) && (len < minLen)) continue;
00182             else if (next < 0) break;
00183             cur = next;
00184             res[bufferctr + len++] = cur;
00185         }
00186         res[bufferctr + len++] = '\n';
00187         bufferctr+=len;
00188
00189     }
00190     if(bFileIO){
00191         mlock->lock();
00192         *wordlist << res;
00193         mlock->unlock();
00194     }else{
00195         mlock->lock();
00196         std::cout << res;
00197         mlock->unlock();
00198     }
00199     delete res;
00200
00201 }
```

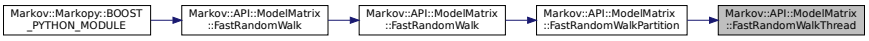
References edgeMatrix, matrixIndex, matrixSize, Markov::Random::Marsaglia::random(), totalEdgeWeights, and valueMatrix.

Referenced by FastRandomWalkPartition().

Here is the call graph for this function:



Here is the caller graph for this function:



8.28.3.13 Generate()

```
void Markov::API::MarkovPasswords::Generate (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20 )  [inherited]
```

Call Markov::Model::RandomWalk n times, and collect output.
Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See Markov::API::MatrixModel::FastRandomWalk for more information.

Parameters

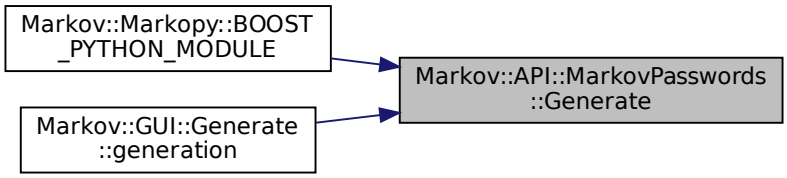
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

Definition at line 118 of file markovPasswords.cpp.

```
00118
                                {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
      &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

References Markov::API::MarkovPasswords::GenerateThread().
Referenced by Markov::Markopy::BOOST_PYTHON_MODULE(), and Markov::GUI::Generate::generation().
Here is the call graph for this function:

Here is the caller graph for this function:



8.28.3.14 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
            std::mutex * outputLock,
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen,
            int maxLen )  [private], [inherited]
```

A single thread invoked by the Generate function.
DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Parameters

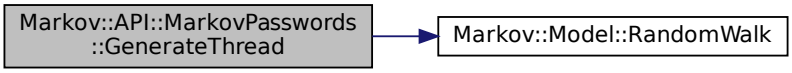
outputLock	- shared mutex lock to lock during output operation. Prevents race condition on write.
n	number of lines to be generated by this thread
wordlist	wordlistfile
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

Definition at line 140 of file markovPasswords.cpp.

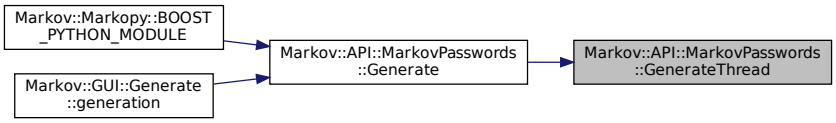
```
00140                                       {
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist « res « "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

References Markov::Model< NodeStorageType >::RandomWalk().
Referenced by Markov::API::MarkovPasswords::Generate().

Here is the call graph for this function:



Here is the caller graph for this function:



8.28.3.15 Import() [1/2]

```
void Markov::API::ModelMatrix::Import (
            const char * filename )
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.

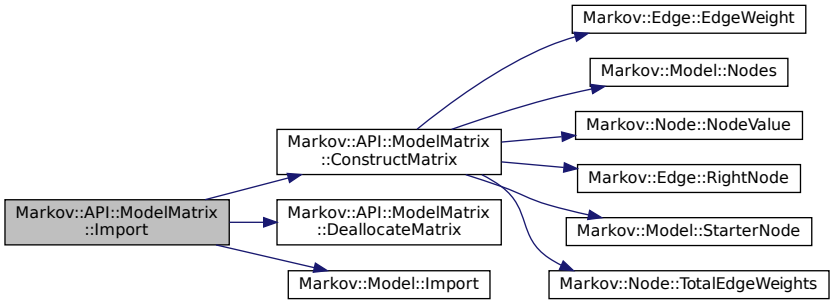
Definition at line 19 of file modelMatrix.cpp.

```
00019                                                   {
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

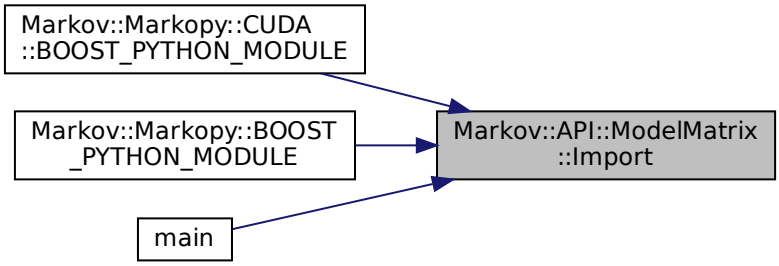
References ConstructMatrix(), DeallocateMatrix(), and Markov::Model< NodeStorageType >::Import().

Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



8.28.3.16 Import() [2/2]

```
bool Markov::Model< char >::Import (
            std::ifstream * f )  [inherited]
```

Import a file to construct the model.
File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr
Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file model.h.

```
00216                                                    {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout « "cell: " « cell « std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout « oc « "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout « "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout « "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
```

```
00252         this->edges.push_back(e);
00253
00254         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

8.28.3.17 Nodes()

`std::map<char , Node<char >*>* Markov::Model< char >::Nodes ( )  [inline], [inherited]`

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.28.3.18 OpenDatasetFile()

`std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (`
`const char * filename )  [inherited]`

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

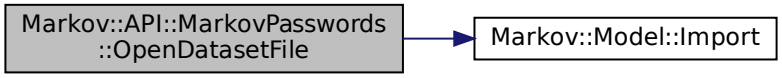
ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References Markov::Model< NodeStorageType >::Import().

Here is the call graph for this function:



8.28.3.19 OptimizeEdgeOrder()

```
void Markov::Model< char >::OptimizeEdgeOrder  [inherited]
```

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.28.3.20 RandomWalk()

```
char * Markov::Model< char >::RandomWalk (
            Markov::Random::RandomEngine * randomEngine,
            int minSetting,
            int maxSetting,
            NodeStorageType * buffer )  [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length 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 86 of file model.h.

```
00307                                                        {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
```

```
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319 
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323 
00324         n = temp_node;
00325 
00326         buffer[len++] = n->NodeValue();
00327     }
00328 
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331 
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.28.3.21 Save()

```
std::ofstream * Markov::API::MarkovPasswords::Save (
            const char * filename )  [inherited]
```

Export model to file.

Parameters

filename	- Export filename.

Returns

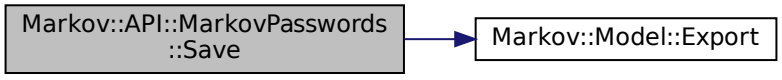
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                   {
00107     std::ofstream* exportFile;
00108 
00109     std::ofstream newFile(filename);
00110 
00111     exportFile = &newFile;
00112 
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References Markov::Model< NodeStorageType >::Export().

Here is the call graph for this function:



8.28.3.22 StarterNode()

```
Node<char >* Markov::Model< char >::StarterNode ( )  [inline], [inherited]
```

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.28.3.23 Train()

```
void Markov::API::ModelMatrix::Train (
            const char * datasetFileName,
            char delimiter,
            int threads )
```

Train the model with the dataset file.

Parameters

datasetFileName	- Ifstream* to the dataset. If null, use class member
delimiter	- a character, same as the delimiter in dataset content
threads	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

Construct the matrix when done.

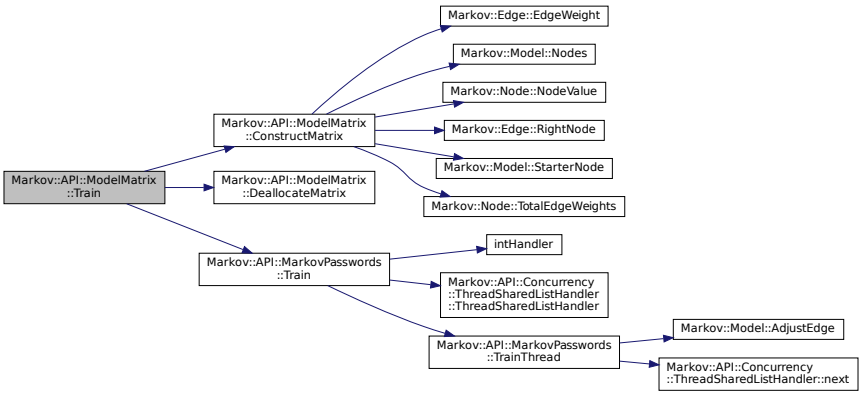
Definition at line 25 of file modelMatrix.cpp.

```
00025                                                                                                {
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

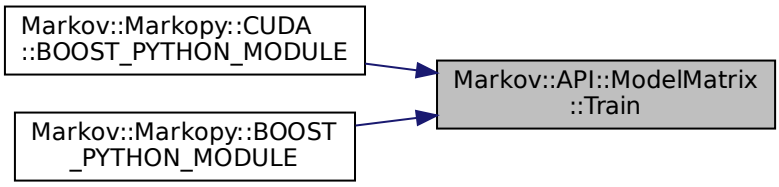
References ConstructMatrix(), DeallocateMatrix(), and Markov::API::MarkovPasswords::Train().

Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), and Markov::Markopy::BOOST_PYTHON_MODULE().

Here is the call graph for this function:



Here is the caller graph for this function:



8.28.3.24 TrainThread()

```
void Markov::API::MarkovPasswords::TrainThread (
            Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
            char delimiter )  [private], [inherited]
```

A single thread invoked by the Train function.

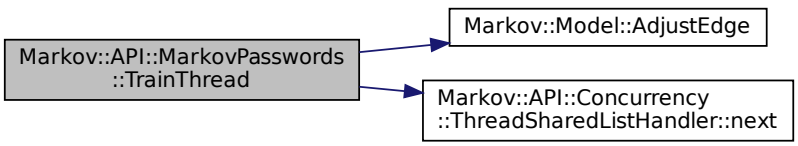
Parameters

listhandler	- Listhandler class to read corpus from
delimiter	- a character, same as the delimiter in dataset content

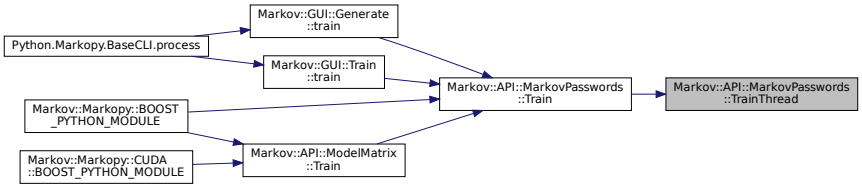
Definition at line 85 of file markovPasswords.cpp.

```
00085                 {
00086     char format_str[] ="%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
      "%ld,%s"
00097 #else
00098         sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100         this->AdjustEdge((const char*)linebuf, oc);
00101         delete linebuf;
00102     }
00103 }
```

References Markov::Model< NodeStorageType >::AdjustEdge(), keepRunning, and Markov::API::Concurrency::ThreadSharedListHa
Referenced by Markov::API::MarkovPasswords::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.28.4 Member Data Documentation

8.28.4.1 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]`
Definition at line 123 of file markovPasswords.h.

8.28.4.2 edgeMatrix

`char** Markov::API::ModelMatrix::edgeMatrix [protected]`
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line 175 of file modelMatrix.h.
Referenced by ConstructMatrix(), DeallocateMatrix(), DumpJSON(), and FastRandomWalkThread().

8.28.4.3 edges

`std::vector<Edge<char >*> Markov::Model< char >::edges [private], [inherited]`
A list of all edges in this model.
Definition at line 204 of file model.h.

8.28.4.4 matrixIndex

`char* Markov::API::ModelMatrix::matrixIndex [protected]`
to hold the Matrix index (To hold the orders of 2-D arrays')
Definition at line 190 of file modelMatrix.h.
Referenced by ConstructMatrix(), DeallocateMatrix(), DumpJSON(), and FastRandomWalkThread().

8.28.4.5 matrixSize

`int Markov::API::ModelMatrix::matrixSize [protected]`
to hold Matrix size
Definition at line 185 of file modelMatrix.h.
Referenced by ConstructMatrix(), DeallocateMatrix(), DumpJSON(), FastRandomWalkThread(), and Markov::API::CUDA::CUDAMode

8.28.4.6 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]`
Dataset file input of our system

Definition at line 124 of file markovPasswords.h.

8.28.4.7 nodes

std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.

8.28.4.8 outputFile

std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]
File to save model of our system

Definition at line 125 of file markovPasswords.h.

8.28.4.9 ready

bool Markov::API::ModelMatrix::ready [protected]
True when matrix is constructed. False if not.
Definition at line 200 of file modelMatrix.h.
Referenced by ConstructMatrix(), DeallocateMatrix(), and ModelMatrix().

8.28.4.10 starterNode

Node<char >* Markov::Model< char >::starterNode [private], [inherited]
Starter Node of this model.
Definition at line 198 of file model.h.

8.28.4.11 totalEdgeWeights

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected]
Array of the Total Edge Weights.
Definition at line 195 of file modelMatrix.h.
Referenced by ConstructMatrix(), DeallocateMatrix(), and FastRandomWalkThread().

8.28.4.12 valueMatrix

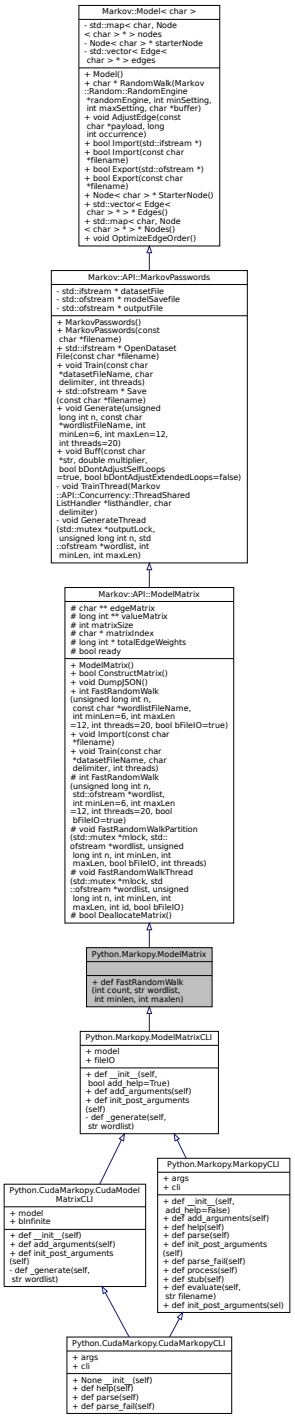
long int** Markov::API::ModelMatrix::valueMatrix [protected]
2-d Integer array for the value Matrix (For the weights of Edges)
Definition at line 180 of file modelMatrix.h.
Referenced by ConstructMatrix(), DeallocateMatrix(), DumpJSON(), and FastRandomWalkThread().
The documentation for this class was generated from the following files:

- Markopy/MarkovAPI/src/modelMatrix.h
- Markopy/MarkovAPI/src/modelMatrix.cpp

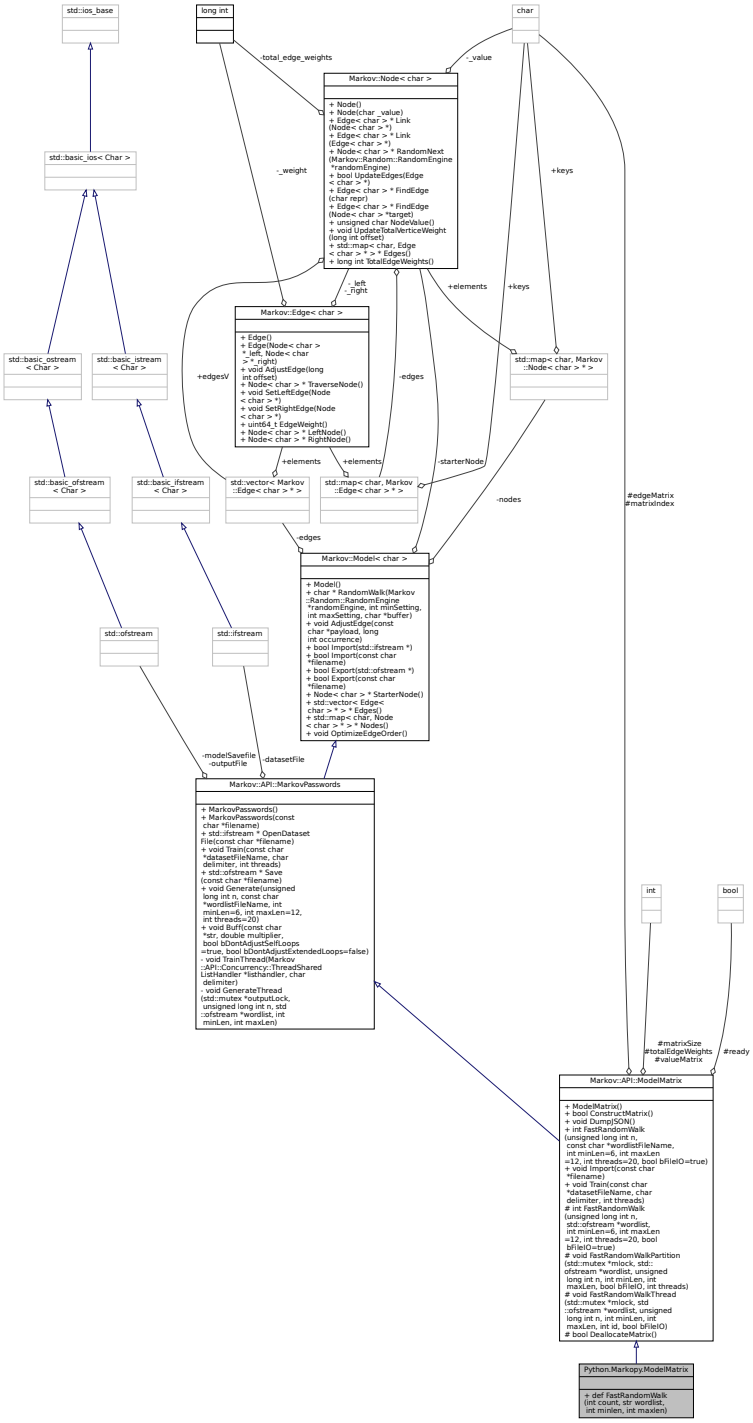
8.29 Python.Markopy.ModelMatrix Class Reference

Abstract representation of a matrix based model.

Inheritance diagram for Python.Markopy.ModelMatrix:



Collaboration diagram for Python.Markopy.ModelMatrix:



Public Member Functions

- def FastRandomWalk (int count, str wordlist, int minlen, int maxlen)
- bool ConstructMatrix ()

  Construct the related Matrix data for the model.
- void DumpJSON ()

  Debug function to dump the model to a JSON file.

- int FastRandomWalk (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- void Import (const char *filename)

  Open a file to import with filename, and call bool Model::Import with std::ifstream.
- bool Import (std::ifstream *)

  Import a file to construct the model.
- void Train (const char *datasetFileName, char delimiter, int threads)

  Train the model with the dataset file.
- std::ifstream * OpenDatasetFile (const char *filename)

  Open dataset file and return the ifstream pointer.
- std::ofstream * Save (const char *filename)

  Export model to file.
- void Generate (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

  Call Markov::Model::RandomWalk n times, and collect output.
- void Buff (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- char * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

  Do a random walk on this model.
- void AdjustEdge (const char *payload, long int occurrence)

  Adjust the model with a single string.
- bool Export (std::ofstream *)

  Export a file of the model.
- bool Export (const char *filename)

  Open a file to export with filename, and call bool Model::Export with std::ofstream.
- Node< char > * StarterNode ()

  Return starter Node.
- std::vector< Edge< char > * > * Edges ()

  Return a vector of all the edges in the model.
- std::map< char, Node< char > * > * Nodes ()

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.

Protected Member Functions

- int FastRandomWalk (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- void FastRandomWalkPartition (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

  A single partition of FastRandomWalk event.
- void FastRandomWalkThread (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)

  A single thread of a single partition of FastRandomWalk.
- bool DeallocateMatrix ()

  Deallocate matrix and make it ready for re-construction.

Protected Attributes

- char ** edgeMatrix

  2-D Character array for the edge Matrix (The characters of Nodes)

- long int ** valueMatrix

  2-d Integer array for the value Matrix (For the weights of Edges)

- int matrixSize

  to hold Matrix size

- char * matrixIndex

  to hold the Matrix index (To hold the orders of 2-D arrays')

- long int * totalEdgeWeights

  Array of the Total Edge Weights.

- bool ready

  True when matrix is constructed. False if not.

Private Member Functions

- void TrainThread (Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char delimiter)

  A single thread invoked by the Train function.

- void GenerateThread (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)

  A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * datasetFile
- std::ofstream * modelSavefile

  Dataset file input of our system

- std::ofstream * outputFile

  File to save model of our system

- std::map< char, Node< char > * > nodes

  Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.

- Node< char > * starterNode

  Starter Node of this model.

- std::vector< Edge< char > * > edges

  A list of all edges in this model.

8.29.1 Detailed Description

Abstract representation of a matrix based model.
To help with the python-cpp gateway documentation.
Definition at line 38 of file mm.py.

8.29.2 Member Function Documentation

8.29.2.1 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
            const NodeStorageType * payload,
            long int occurrence )  [inherited]
```

Adjust the model with a single string.
Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.
Then, update the edge EdgeWeight from current node, to the terminator node.
This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.
Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file model.h.

```
00337                                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.29.2.2 Buff()

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file markovPasswords.cpp.

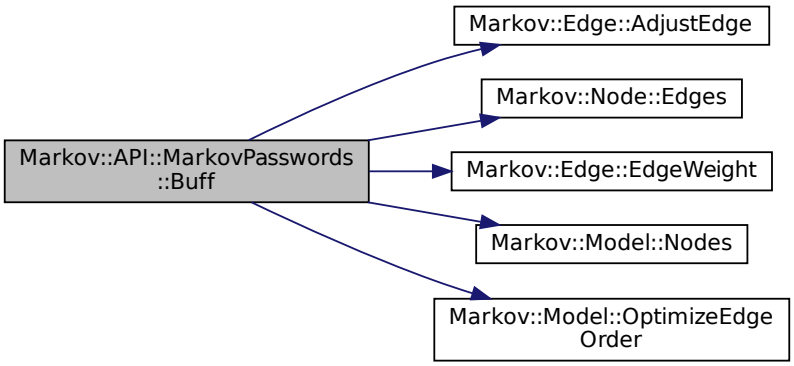
```
00153
                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
```

```
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173 
00174         }
00175         i++;
00176     }
00177 
00178     this->OptimizeEdgeOrder();
00179 }
```

References Markov::Edge< NodeStorageType >::AdjustEdge(), Markov::Node< storageType >::Edges(), Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::Model< NodeStorageType >::Nodes(), and Markov::Model< NodeStorageType >::OptimizeEdgeOrder().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.29.2.3 ConstructMatrix()

```
bool Markov::API::ModelMatrix::ConstructMatrix ( )  [inherited]
```

Construct the related Matrix data for the model.

This operation can be used after importing/training to allocate and populate the matrix content.

this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

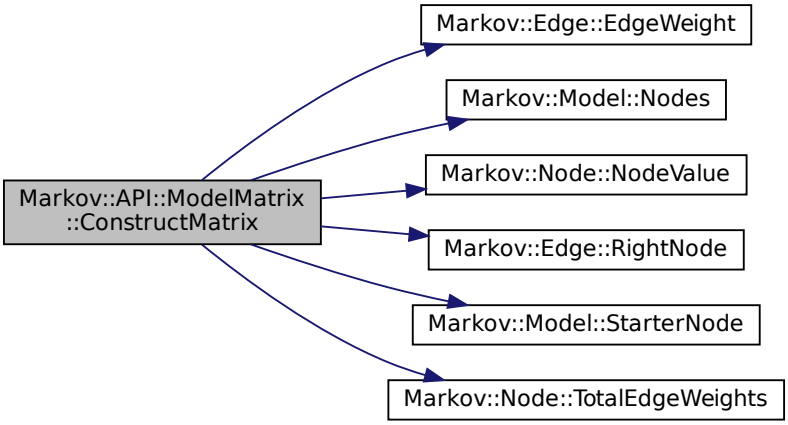
True if constructed. False if already construced.

Definition at line 31 of file modelMatrix.cpp.

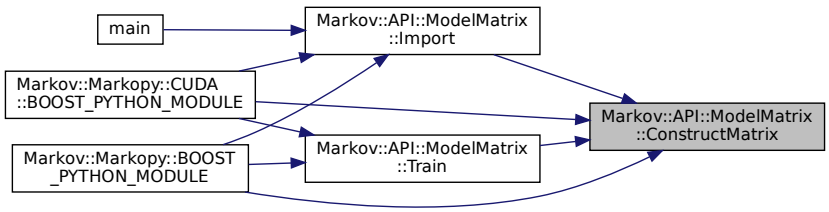
```
00031                                           {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034 
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037 
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072 
00073         }
00074         i++;
00075     }
00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::API::ModelMatrix::r
Markov::API::ModelMatrix::matrixSize, Markov::Model< NodeStorageType >::Nodes(), Markov::Node< storageType >::NodeValue()
Markov::API::ModelMatrix::ready, Markov::Edge< NodeStorageType >::RightNode(), Markov::Model< NodeStorageType >::StarterN
Markov::API::ModelMatrix::totalEdgeWeights, Markov::Node< storageType >::TotalEdgeWeights(), and Markov::API::ModelMatrix::va
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(),
Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().

Here is the call graph for this function:



Here is the caller graph for this function:



8.29.2.4 DeallocateMatrix()

`bool Markov::API::ModelMatrix::DeallocateMatrix ( )  [protected], [inherited]`

Deallocate matrix and make it ready for re-construction.

Returns

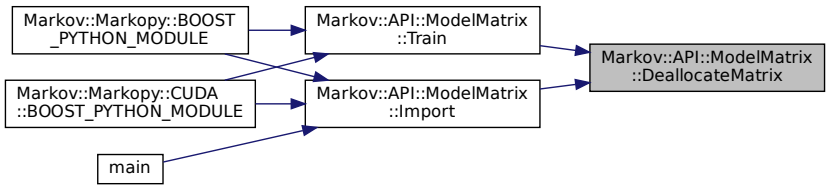
True if deallocated. False if matrix was not initialized

Definition at line 81 of file modelMatrix.cpp.

```
00081                                                {
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
00085 
00086     for(int i=0;i<this->matrixSize;i++){
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090 
00091     for(int i=0;i<this->matrixSize;i++){
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095 
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
```

```
00099 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, Markov::API::ModelMatrix::ready, Markov::API::ModelMatrix::totalEdgeWeights, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().
Here is the caller graph for this function:



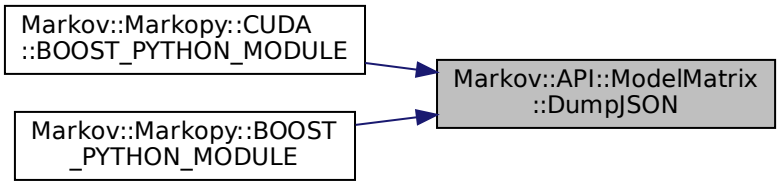
8.29.2.5 DumpJSON()

```
void Markov::API::ModelMatrix::DumpJSON ( )  [inherited]
```

Debug function to dump the model to a JSON file.
Might not work 100%. Not meant for production use.
Definition at line 101 of file modelMatrix.cpp.

```
00101                                   {
00102
00103     std::cout « "{\n   \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout « "\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout « "\\\\";
00107         else if(this->matrixIndex[i]==0) std::cout « "\\\\x00";
00108         else if(i==0) std::cout « "\\\\xff";
00109         else if(this->matrixIndex[i]=='\n') std::cout « "\\n";
00110         else std::cout « this->matrixIndex[i];
00111     }
00112     std::cout «
00113     "\",\n"
00114     "   \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++){
00117         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00120         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00121         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00122         for(int j=0;j<this->matrixSize;j++){
00123             if(this->edgeMatrix[i][j]=='"') std::cout « "\"\\\"\"";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout « "\"\\\\\"";
00125             else if(this->edgeMatrix[i][j]==0) std::cout « "\"\\\\x00\"";
00126             else if(this->edgeMatrix[i][j]<0) std::cout « "\"\\\\xff\"";
00127             else if(this->matrixIndex[i]=='\n') std::cout « "\"\\n\"";
00128             else std::cout « "\"" « this->edgeMatrix[i][j] « "\"";
00129             if(j!=this->matrixSize-1) std::cout « ", ";
00130         }
00131         std::cout « "],\n";
00132     }
00133     std::cout « "},\n";
00134
00135     std::cout « "\"   weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++){
00137         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00140         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00141         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++){
00144             std::cout « this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout « ", ";
00146         }
00147         std::cout « "],\n";
00148     }
00149     std::cout « "  }\n}\n";
00150 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, and Markov::API::ModelMatrix::valueMatrix.
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), and Markov::Markopy::BOOST_PYTHON_MODULE().
Here is the caller graph for this function:



8.29.2.6 Edges()

```
std::vector<Edge<char >*>* Markov::Model< char >::Edges ( )  [inline], [inherited]
```

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.29.2.7 Export() [1/2]

```
bool Markov::Model< char >::Export (
            const char * filename )  [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file model.h.

```
00300                                                    {
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

8.29.2.8 Export() [2/2]

```
bool Markov::Model< char >::Export (
            std::ofstream * f )  [inherited]
```

Export a file of the model.
File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.
Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file model.h.

```
00288                                                   {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
      e->RightNode()->NodeValue() « "\n";
00293         *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.29.2.9 FastRandomWalk() [1/3]

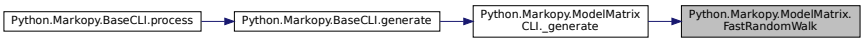
```
def Python.Markopy.ModelMatrix.FastRandomWalk (
            int count,
            str wordlist,
            int minlen,
            int maxlen )
```

Definition at line 48 of file mm.py.

```
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass
```

Referenced by Python.Markopy.ModelMatrixCLI._generate().

Here is the caller graph for this function:



8.29.2.10 FastRandomWalk() [2/3]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [inherited]
```

Random walk on the Matrix-reduced Markov::Model.

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5

This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to

Parameters

minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 217 of file modelMatrix.cpp.

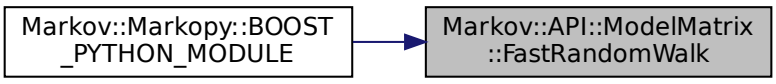
```
00217                                                {
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

References Markov::API::ModelMatrix::FastRandomWalk().

Referenced by Markov::Markopy::BOOST_PYTHON_MODULE().

Here is the call graph for this function:

Here is the caller graph for this function:



8.29.2.11 FastRandomWalk() [3/3]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [protected], [inherited]
```

Random walk on the Matrix-reduced Markov::Model.

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5

This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

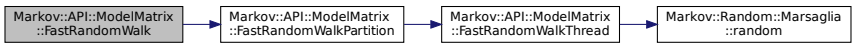
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 204 of file modelMatrix.cpp.

```
00204
                                    {
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
      threads);
00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
      threads);
00213     }
00214     return 0;
00215 }
```

References Markov::API::ModelMatrix::FastRandomWalkPartition().
Referenced by Markov::API::ModelMatrix::FastRandomWalk().
Here is the call graph for this function:



Here is the caller graph for this function:

8.29.2.12 FastRandomWalkPartition()

```
void Markov::API::ModelMatrix::FastRandomWalkPartition (
            std::mutex * mlock,
            std::ofstream * wordlist,
            unsigned long int n,
            int minLen,
            int maxLen,
            bool bFileIO,
            int threads )  [protected], [inherited]
```

A single partition of FastRandomWalk event.
Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation

- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

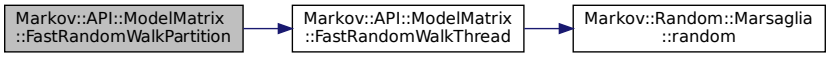
Parameters

mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

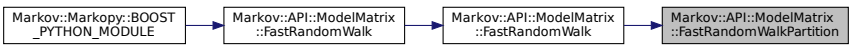
Definition at line 225 of file modelMatrix.cpp.

```
00225                                                                    {
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++){
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
      mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
      wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00239
00240     for(int i=0;i<threads;i++){
00241         threadsV[i]->join();
00242     }
00243 }
```

References Markov::API::ModelMatrix::FastRandomWalkThread().
Referenced by Markov::API::ModelMatrix::FastRandomWalk().
Here is the call graph for this function:



Here is the caller graph for this function:



8.29.2.13 FastRandomWalkThread()

```
void Markov::API::ModelMatrix::FastRandomWalkThread (
            std::mutex * mlock,
```

```
            std::ofstream * wordlist,
            unsigned long int n,
            int minLen,
            int maxLen,
            int id,
            bool bFileIO )  [protected], [inherited]
```

A single thread of a single partition of FastRandomWalk.
A FastRandomWalkPartition will initiate as many of this function as requested.
This function contains the bulk of the generation algorithm.

Parameters

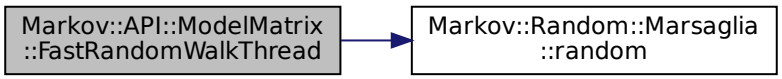
mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
id	- **DEPRECATED** Thread id - No longer used
bFileIO	- If false, filename will be ignored and will output to stdout.

Definition at line 153 of file modelMatrix.cpp.

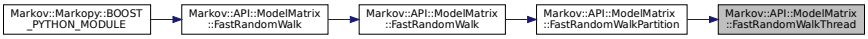
```
00153                                                                   {
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179
00180             if (len >= maxLen)  break;
00181             else if ((next < 0) && (len < minLen)) continue;
00182             else if (next < 0) break;
00183             cur = next;
00184             res[bufferctr + len++] = cur;
00185         }
00186         res[bufferctr + len++] = '\n';
00187         bufferctr+=len;
00188
00189     }
00190     if(bFileIO){
00191         mlock->lock();
00192         *wordlist << res;
00193         mlock->unlock();
00194     }else{
00195         mlock->lock();
00196         std::cout << res;
00197         mlock->unlock();
00198     }
00199     delete res;
00200
00201 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, Markov::Random::Marsaglia::random(), Markov::API::ModelMatrix::totalEdgeWeights, and Markov::API::ModelMatrix::valueMatrix.

Referenced by Markov::API::ModelMatrix::FastRandomWalkPartition().
Here is the call graph for this function:



Here is the caller graph for this function:



8.29.2.14 Generate()

```
void Markov::API::MarkovPasswords::Generate (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20 )  [inherited]
```

Call Markov::Model::RandomWalk n times, and collect output.
Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See Markov::API::MatrixModel::FastRandomWalk for more information.

Parameters

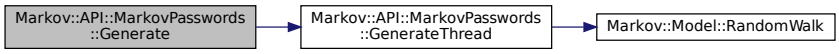
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

Definition at line 118 of file markovPasswords.cpp.

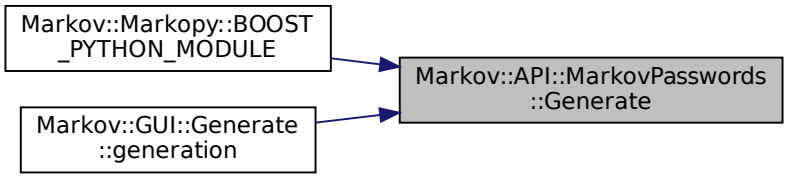
```
00118                                {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
       &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
```

```
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

References Markov::API::MarkovPasswords::GenerateThread().
Referenced by Markov::Markopy::BOOST_PYTHON_MODULE(), and Markov::GUI::Generate::generation().
Here is the call graph for this function:



Here is the caller graph for this function:



8.29.2.15 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
            std::mutex * outputLock,
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen,
            int maxLen )  [private], [inherited]
```

A single thread invoked by the Generate function.
DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Parameters

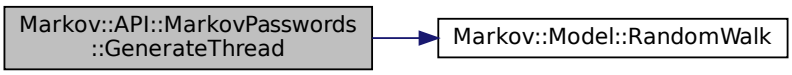
outputLock	- shared mutex lock to lock during output operation. Prevents race condition on write.
n	number of lines to be generated by this thread
wordlist	wordlistfile
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

Definition at line 140 of file markovPasswords.cpp.

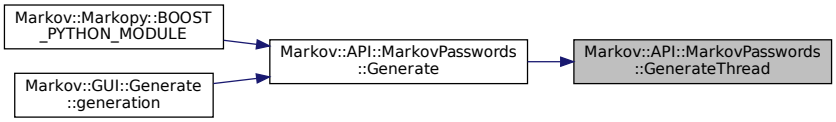
```
00140                                       {
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
```

```
00148         *wordlist « res « "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

References Markov::Model< NodeStorageType >::RandomWalk().
Referenced by Markov::API::MarkovPasswords::Generate().
Here is the call graph for this function:



Here is the caller graph for this function:



8.29.2.16 Import() [1/2]

```
void Markov::API::ModelMatrix::Import (
            const char * filename )  [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

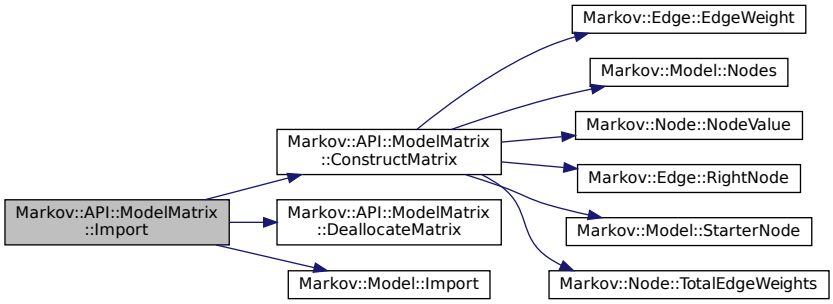
```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.
Definition at line 19 of file modelMatrix.cpp.

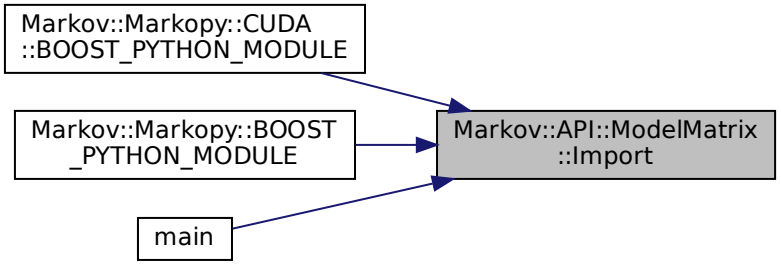
```
00019                                                   {
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::Model< NodeStorageType >::Import().
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



8.29.2.17 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 126 of file model.h.

```
00216                                                    {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout « "cell: " « cell « std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
```

```
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout << "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
      int(targetN->NodeValue()) << "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

8.29.2.18 Nodes()

`std::map<char , Node<char >*>* Markov::Model< char >::Nodes ( ) [inline], [inherited]`

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.29.2.19 OpenDatasetFile()

`std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile ( const char * filename ) [inherited]`

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

ifstream* to the the dataset file

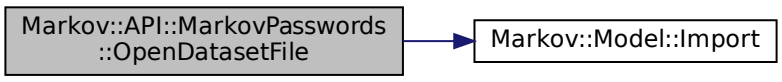
Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                    {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
```

```
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References Markov::Model< NodeStorageType >::Import().

Here is the call graph for this function:



8.29.2.20 OptimizeEdgeOrder()

```
void Markov::Model< char >::OptimizeEdgeOrder  [inherited]
```

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.29.2.21 RandomWalk()

```
char * Markov::Model< char >::RandomWalk (
            Markov::Random::RandomEngine * randomEngine,
            int minSetting,
            int maxSetting,
            NodeStorageType * buffer )  [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criateria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes Markov::Random::RandomEngine as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so Markov::Random::Marsaglia is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout « res « "\n";
 }
```

Parameters

randomEngine	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
minSetting	Minimum number of characters to generate
maxSetting	Maximum number of character to generate
buffer	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307                                                                {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue();
00327     }
00328
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.29.2.22 Save()

```
std::ofstream * Markov::API::MarkovPasswords::Save (
            const char * filename )  [inherited]
```

Export model to file.

Parameters

filename	- Export filename.

Returns

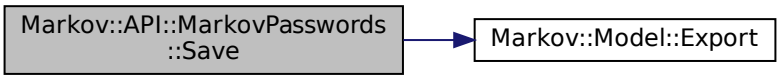
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References Markov::Model< NodeStorageType >::Export().

Here is the call graph for this function:



8.29.2.23 StarterNode()

```
Node<char >* Markov::Model< char >::StarterNode ( )  [inline], [inherited]
```

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.29.2.24 Train()

```
void Markov::API::ModelMatrix::Train (
            const char * datasetFileName,
            char delimiter,
            int threads )  [inherited]
```

Train the model with the dataset file.

Parameters

datasetFileName	- Ifstream* to the dataset. If null, use class member
delimiter	- a character, same as the delimiter in dataset content
threads	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

Construct the matrix when done.

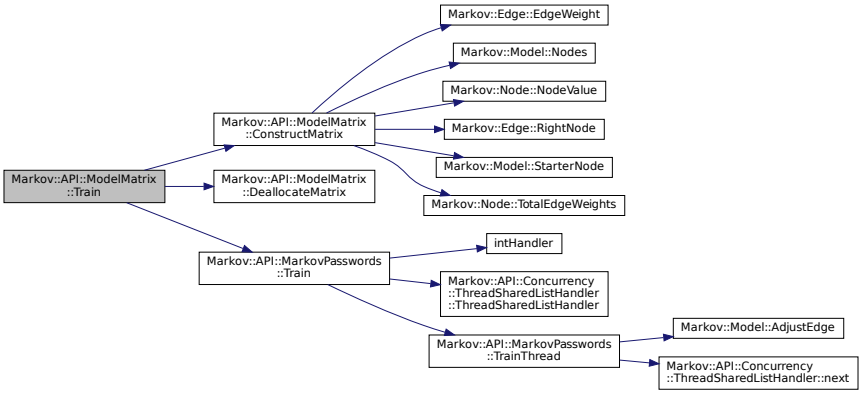
Definition at line 25 of file modelMatrix.cpp.

```
00025                                                                                {
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

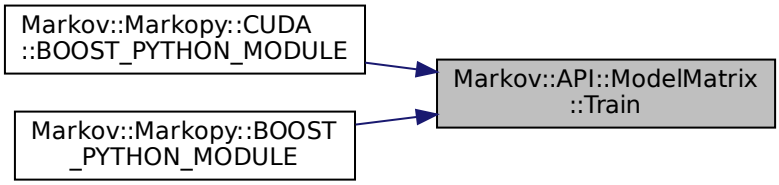
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::MarkovPasswords::Train().

Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), and Markov::Markopy::BOOST_PYTHON_MODULE().

Here is the call graph for this function:



Here is the caller graph for this function:



8.29.2.25 TrainThread()

```
void Markov::API::MarkovPasswords::TrainThread (
            Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
            char delimiter )  [private], [inherited]
```

A single thread invoked by the Train function.

Parameters

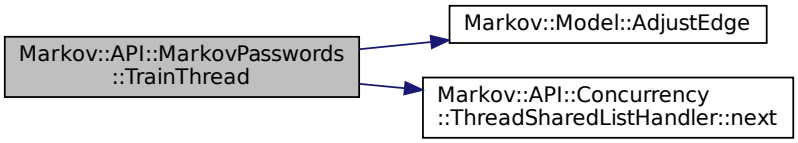
listhandler	- Listhandler class to read corpus from
delimiter	- a character, same as the delimiter in dataset content

Definition at line 85 of file markovPasswords.cpp.

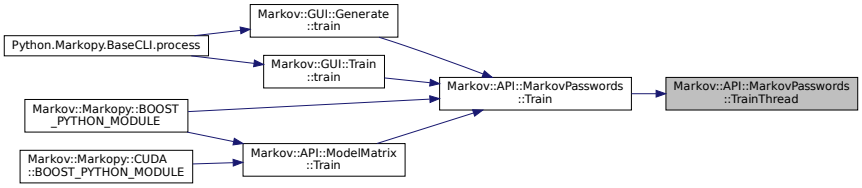
```
00085
                {
00086     char format_str[] ="%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
      "%ld,%s"
00097 #else
```

```
00098         sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100         this->AdjustEdge((const char*)linebuf, oc);
00101         delete linebuf;
00102     }
00103 }
```

References Markov::Model< NodeStorageType >::AdjustEdge(), keepRunning, and Markov::API::Concurrency::ThreadSharedListHa
Referenced by Markov::API::MarkovPasswords::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.29.3 Member Data Documentation

8.29.3.1 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]`
Definition at line 123 of file markovPasswords.h.

8.29.3.2 edgeMatrix

`char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]`
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line 175 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.29.3.3 edges

`std::vector<Edge<char >*> Markov::Model< char >::edges [private], [inherited]`
A list of all edges in this model.
Definition at line 204 of file model.h.

8.29.3.4 matrixIndex

`char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]`
to hold the Matrix index (To hold the orders of 2-D arrays')
Definition at line 190 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.29.3.5 matrixSize

`int Markov::API::ModelMatrix::matrixSize [protected], [inherited]`
to hold Matrix size
Definition at line 185 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), Markov::API::ModelMatrix::FastRandomWalkThread(), and Markov::API::CUDA::CUDAMod

8.29.3.6 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]`
Dataset file input of our system

Definition at line 124 of file markovPasswords.h.

8.29.3.7 nodes

`std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]`
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.

8.29.3.8 outputFile

`std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]`
File to save model of our system

Definition at line 125 of file markovPasswords.h.

8.29.3.9 ready

`bool Markov::API::ModelMatrix::ready [protected], [inherited]`
True when matrix is constructed. False if not.
Definition at line 200 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::ModelMatrix().

8.29.3.10 starterNode

`Node<char >* Markov::Model< char >::starterNode [private], [inherited]`
Starter Node of this model.
Definition at line 198 of file model.h.

8.29.3.11 totalEdgeWeights

```
long int* Markov::API::ModelMatrix::totalEdgeWeights  [protected], [inherited]
```

Array of the Total Edge Weights.
Definition at line 195 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.29.3.12 valueMatrix

```
long int** Markov::API::ModelMatrix::valueMatrix  [protected], [inherited]
```

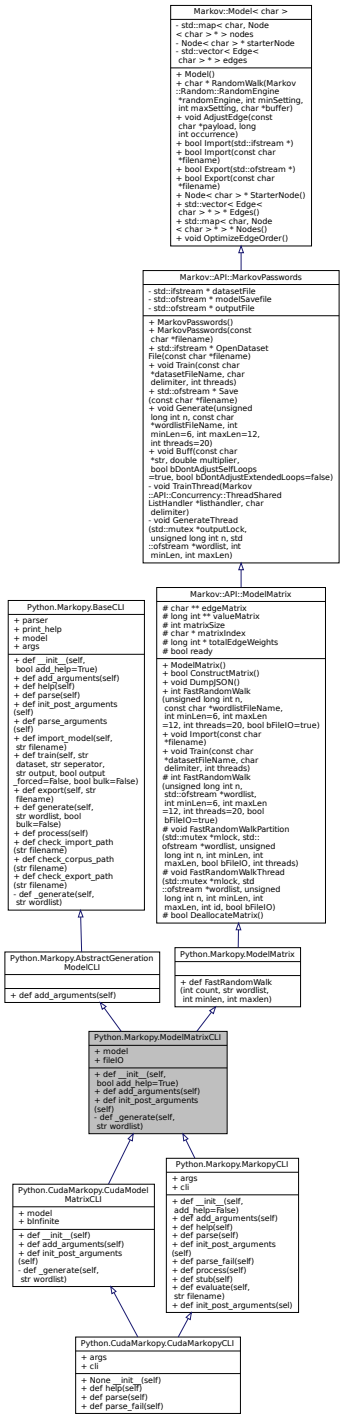
2-d Integer array for the value Matrix (For the weights of Edges)
Definition at line 180 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().
The documentation for this class was generated from the following file:

- Markopy/Markopy/src/CLI/mm.py

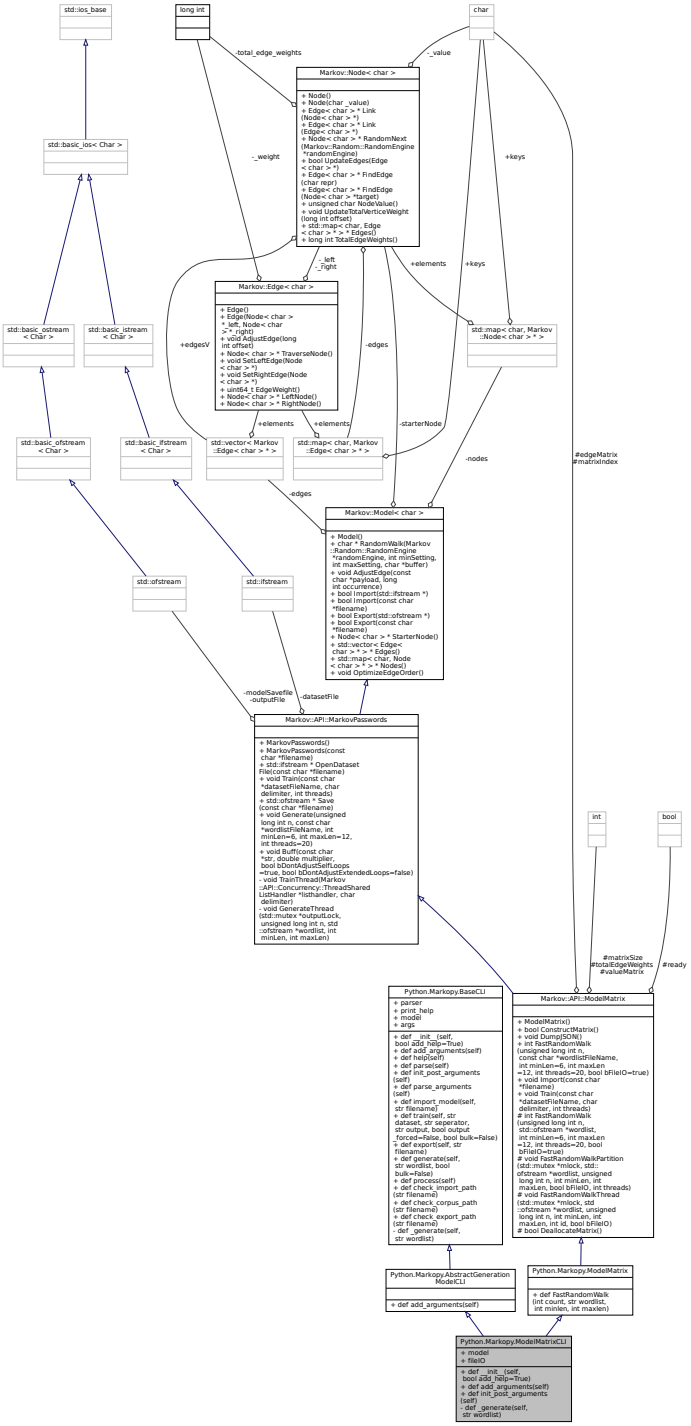
8.30 Python.Markopy.ModelMatrixCLI Class Reference

Extension of Python.Markopy.Base.BaseCLI for Markov::API::ModelMatrix.

Inheritance diagram for Python.Markopy.ModelMatrixCLI:



Collaboration diagram for Python.Markopy.ModelMatrixCLI:



Public Member Functions

- def __init__ (self, bool add_help=True)

  initialize base CLI
- def add_arguments (self)
- def init_post_arguments (self)
- def help (self)
- def parse (self)

- def parse_arguments (self)
- def import_model (self, str filename)

  Import a model file.
- def train (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

  Train a model via CLI parameters.
- def export (self, str filename)

  Export model to a file.
- def generate (self, str wordlist, bool bulk=False)

  Generate strings from the model.
- def process (self)

  Process parameters for operation.
- def FastRandomWalk (int count, str wordlist, int minlen, int maxlen)
- int FastRandomWalk (unsigned long int n, const char ∗wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- bool ConstructMatrix ()

  Construct the related Matrix data for the model.
- void DumpJSON ()

  Debug function to dump the model to a JSON file.
- void Import (const char ∗filename)

  Open a file to import with filename, and call bool Model::Import with std::ifstream.
- bool Import (std::ifstream ∗)

  Import a file to construct the model.
- void Train (const char ∗datasetFileName, char delimiter, int threads)

  Train the model with the dataset file.
- std::ifstream ∗ OpenDatasetFile (const char ∗filename)

  Open dataset file and return the ifstream pointer.
- std::ofstream ∗ Save (const char ∗filename)

  Export model to file.
- void Generate (unsigned long int n, const char ∗wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

  Call Markov::Model::RandomWalk n times, and collect output.
- void Buff (const char ∗str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtended↩ Loops=false)

  Buff expression of some characters in the model.
- char ∗ RandomWalk (Markov::Random::RandomEngine ∗randomEngine, int minSetting, int maxSetting, char ∗buffer)

  Do a random walk on this model.
- void AdjustEdge (const char ∗payload, long int occurrence)

  Adjust the model with a single string.
- bool Export (std::ofstream ∗)

  Export a file of the model.
- bool Export (const char ∗filename)

  Open a file to export with filename, and call bool Model::Export with std::ofstream.
- Node< char > ∗ StarterNode ()

  Return starter Node.
- std::vector< Edge< char > ∗ > ∗ Edges ()

  Return a vector of all the edges in the model.
- std::map< char, Node< char > ∗ > ∗ Nodes ()

  Return starter Node.
- void OptimizeEdgeOrder ()

  Sort edges of all nodes in the model ordered by edge weights.

Static Public Member Functions

- def check_import_path (str filename)

  check import path for validity
- def check_corpus_path (str filename)

  check import path for validity
- def check_export_path (str filename)

  check import path for validity

Public Attributes

- model
- fileIO
- parser
- print_help
- args

Protected Member Functions

- int FastRandomWalk (unsigned long int n, std::ofstream ∗wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

  Random walk on the Matrix-reduced Markov::Model.
- void FastRandomWalkPartition (std::mutex ∗mlock, std::ofstream ∗wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

  A single partition of FastRandomWalk event.
- void FastRandomWalkThread (std::mutex ∗mlock, std::ofstream ∗wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)

  A single thread of a single partition of FastRandomWalk.
- bool DeallocateMatrix ()

  Deallocate matrix and make it ready for re-construction.

Protected Attributes

- char ∗∗ edgeMatrix

  2-D Character array for the edge Matrix (The characters of Nodes)
- long int ∗∗ valueMatrix

  2-d Integer array for the value Matrix (For the weights of Edges)
- int matrixSize

  to hold Matrix size
- char ∗ matrixIndex

  to hold the Matrix index (To hold the orders of 2-D arrays')
- long int ∗ totalEdgeWeights

  Array of the Total Edge Weights.
- bool ready

  True when matrix is constructed. False if not.

Private Member Functions

- def _generate (self, str wordlist)

  wrapper for generate function.
- 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

  Dataset file input of our system

- std::ofstream ∗ outputFile

  File to save model of our system

- std::map< char, Node< char > ∗ > nodes

  Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > ∗ starterNode

  Starter Node of this model.
- std::vector< Edge< char > ∗ > edges

  A list of all edges in this model.

8.30.1 Detailed Description

Extension of Python.Markopy.Base.BaseCLI for Markov::API::ModelMatrix.
adds -st/–stdout arguement to the command line.
Definition at line 18 of file mmx.py.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 __init__()

```
def Python.Markopy.ModelMatrixCLI.__init__ (
              self,
             bool add_help = True )
```

initialize base CLI

Parameters

add_help	decide to overload the help function or not

Reimplemented from Python.Markopy.BaseCLI.
Definition at line 27 of file mmx.py.

```
00027     def __init__(self, add_help:bool=True):
00028         "! @brief initialize model with Markov::API::ModelMatrix"
00029         super().__init__(add_help)
00030         self.model = markopy.ModelMatrix()
00031
```

8.30.3 Member Function Documentation

8.30.3.1 _generate()

```
def Python.Markopy.ModelMatrixCLI._generate (
              self,
             str wordlist )  [private]
```

wrapper for generate function.
This can be overloaded by other models

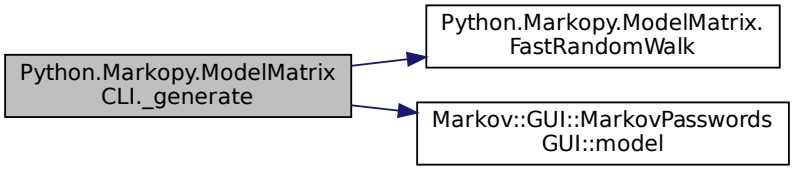
Parameters

wordlist	filename to generate to

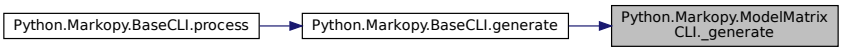
Reimplemented from Python.Markopy.BaseCLI.
Reimplemented in Python.CudaMarkopy.CudaModelMatrixCLI.
Definition at line 40 of file mmx.py.

```
00040     def _generate(self, wordlist : str, ):
00041         self.model.FastRandomWalk(int(self.args.count), wordlist, int(self.args.min),
      int(self.args.max), int(self.args.threads), self.fileIO)
00042
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.ModelMatrix.FastRandomWalk(), Python.Markopy.ModelMatrixCLI.fileIO, Python.CudaMarkopy.CudaModelMatrixCL Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.generate().
Here is the call graph for this function:



Here is the caller graph for this function:



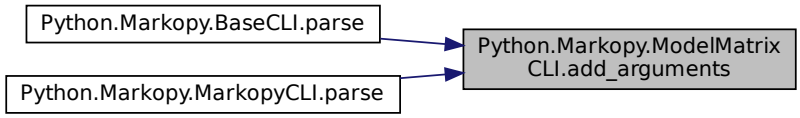
8.30.3.2 add_arguments()

```
def Python.Markopy.ModelMatrixCLI.add_arguments (
              self )
```

Reimplemented from Python.Markopy.AbstractGenerationModelCLI.
Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaModelMatrixCLI.
Definition at line 32 of file mmx.py.

```
00032     def add_arguments(self):
00033         super().add_arguments()
00034         self.parser.add_argument("-st", "--stdout", action="store_true", help="Stdout mode")
00035
```

References Python.Markopy.BaseCLI.parser.
Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().
Here is the caller graph for this function:



8.30.3.3 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
            const NodeStorageType * payload,
            long int occurrence )  [inherited]
```

Adjust the model with a single string.
Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.
Then, update the edge EdgeWeight from current node, to the terminator node.
This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.
Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file model.h.

```
00337                                                                                                {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

8.30.3.4 Buff()

```
void Markov::API::MarkovPasswords::Buff (
            const char * str,
            double multiplier,
            bool bDontAdjustSelfLoops = true,
            bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file markovPasswords.cpp.

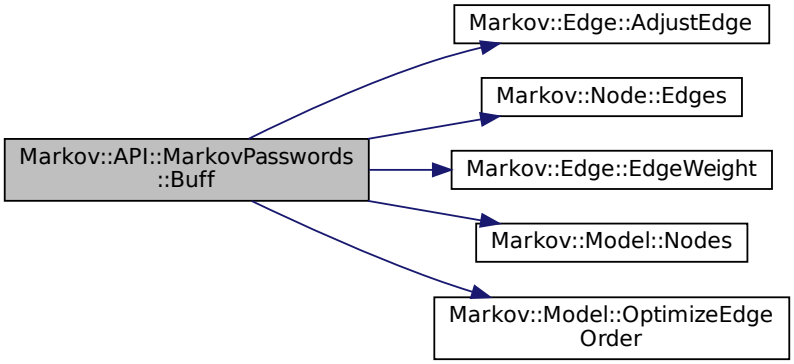
```
00153
                                {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
```

```
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173 
00174         }
00175         i++;
00176     }
00177 
00178     this->OptimizeEdgeOrder();
00179 }
```

References Markov::Edge< NodeStorageType >::AdjustEdge(), Markov::Node< storageType >::Edges(), Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::Model< NodeStorageType >::Nodes(), and Markov::Model< NodeStorageType >::OptimizeEdgeOrder().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

8.30.3.5 check_corpus_path()

```
def Python.Markopy.BaseCLI.check_corpus_path (
            str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 181 of file base.py.

```
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186 
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.30.3.6 check_export_path()

```
def Python.Markopy.BaseCLI.check_export_path (
              str filename )  [static], [inherited]
```

check import path for validity

Parameters

filename	filename to check

Definition at line 192 of file base.py.

```
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197 
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201 
```

Referenced by Python.Markopy.BaseCLI.train().

Here is the caller graph for this function:

8.30.3.7 check_import_path()

```
def Python.Markopy.BaseCLI.check_import_path (
              str filename )  [static], [inherited]
```

check import path for validity

Parameters

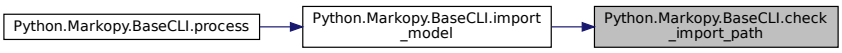
filename	filename to check

Definition at line 169 of file base.py.

```
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
```

Referenced by Python.Markopy.BaseCLI.import_model().

Here is the caller graph for this function:



8.30.3.8 ConstructMatrix()

```
bool Markov::API::ModelMatrix::ConstructMatrix ( )  [inherited]
```

Construct the related Matrix data for the model.

This operation can be used after importing/training to allocate and populate the matrix content.

this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

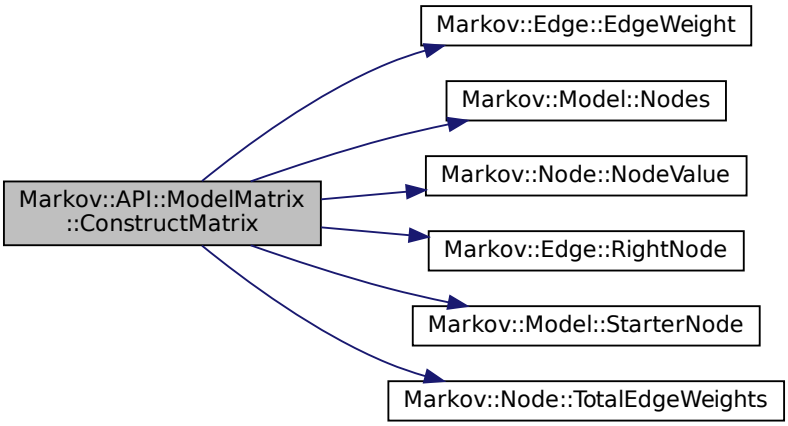
True if constructed. False if already construced.

Definition at line 31 of file modelMatrix.cpp.

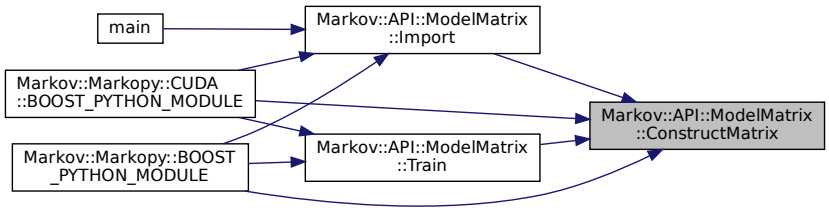
```
00031                                           {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034 
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037 
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
```

```
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072 
00073         }
00074         i++;
00075     }
00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::Edge< NodeStorageType >::EdgeWeight(), Markov::API::ModelMatrix::r
Markov::API::ModelMatrix::matrixSize, Markov::Model< NodeStorageType >::Nodes(), Markov::Node< storageType >::NodeValue()
Markov::API::ModelMatrix::ready, Markov::Edge< NodeStorageType >::RightNode(), Markov::Model< NodeStorageType >::StarterN
Markov::API::ModelMatrix::totalEdgeWeights, Markov::Node< storageType >::TotalEdgeWeights(), and Markov::API::ModelMatrix::va
Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(),
Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.30.3.9 DeallocateMatrix()

```
bool Markov::API::ModelMatrix::DeallocateMatrix ( )  [protected], [inherited]
```

Deallocate matrix and make it ready for re-construction.

Returns

True if deallocated. False if matrix was not initialized

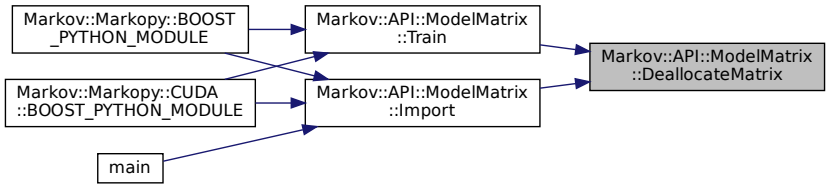
Definition at line 81 of file modelMatrix.cpp.

```
00081                                                {
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
00085 
00086     for(int i=0;i<this->matrixSize;i++){
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090 
00091     for(int i=0;i<this->matrixSize;i++){
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095 
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
00099 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, Markov::API::ModelMatrix::ready, Markov::API::ModelMatrix::totalEdgeWeights, and Markov::API::ModelMatrix::valueMatrix.

Referenced by Markov::API::ModelMatrix::Import(), and Markov::API::ModelMatrix::Train().

Here is the caller graph for this function:



8.30.3.10 DumpJSON()

```
void Markov::API::ModelMatrix::DumpJSON ( )  [inherited]
```

Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.

Definition at line 101 of file modelMatrix.cpp.

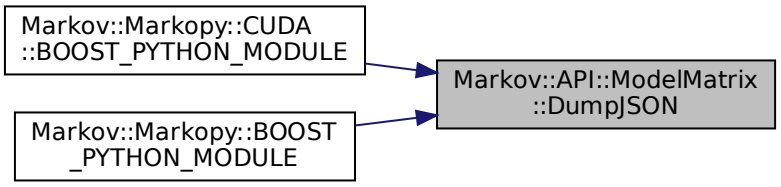
```
00101                                     {
00102 
00103     std::cout << "{\n   \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout << "\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout << "\\\\";
00107         else if(this->matrixIndex[i]==0) std::cout << "\\\\x00";
00108         else if(i==0) std::cout << "\\\\xff";
00109         else if(this->matrixIndex[i]=='\n') std::cout << "\\n";
00110         else std::cout << this->matrixIndex[i];
00111     }
00112     std::cout <<
00113     "\",\n"
00114     "   \"edgemap\": {\n";
00115 
00116     for(int i=0;i<this->matrixSize;i++){
00117         if(this->matrixIndex[i]=='"') std::cout << "      \"\\\"\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\x00\": [";
00120         else if(this->matrixIndex[i]<0) std::cout << "      \"\\\\xff\": [";
00121         else std::cout << "      \"" << this->matrixIndex[i] << "\": [";
00122         for(int j=0;j<this->matrixSize;j++){
00123             if(this->edgeMatrix[i][j]=='"') std::cout << "\"\\\"\"";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout << "\"\\\\\"";
00125             else if(this->edgeMatrix[i][j]==0) std::cout << "\"\\\\x00\"";
00126             else if(this->edgeMatrix[i][j]<0) std::cout << "\"\\\\xff\"";
00127             else if(this->matrixIndex[i]=='\n') std::cout << "\"\\n\"";
```

```
00128             else std::cout « "\"" « this->edgeMatrix[i][j] « "\"";
00129             if(j!=this->matrixSize-1) std::cout « ", ";
00130         }
00131         std::cout « "],\n";
00132     }
00133     std::cout « "},\n";
00134
00135     std::cout « "\"   weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++){
00137         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00140         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00141         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++){
00144             std::cout « this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout « ", ";
00146         }
00147         std::cout « "],\n";
00148     }
00149     std::cout « "  }\n}\n";
00150 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, and Markov::API::ModelMatrix::valueMatrix.

Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), and Markov::Markopy::BOOST_PYTHON_MODULE().

Here is the caller graph for this function:



8.30.3.11 Edges()

```
std::vector<Edge<char >*>* Markov::Model< char >::Edges ( )  [inline], [inherited]
```

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file model.h.

```
00176 { return &edges;}
```

8.30.3.12 Export() [1/2]

```
bool Markov::Model< char >::Export (
            const char * filename )  [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file model.h.

```
00300                                                                    {
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

8.30.3.13 export()

```
def Python.Markopy.BaseCLI.export (
              self,
             str filename )  [inherited]
```

Export model to a file.

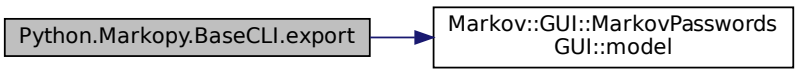
Parameters

filename	filename to export to

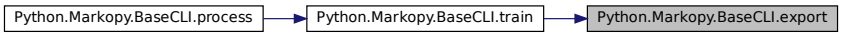
Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.mo
Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.30.3.14 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 155 of file model.h.

```
00288                                                   {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," «
      e->RightNode()->NodeValue() « "\n";
00293         *f « e->LeftNode()->NodeValue() « "," « e->EdgeWeight() « "," « e->RightNode()->NodeValue() «
      "\n";
00294     }
00295
00296     return true;
00297 }
```

8.30.3.15 FastRandomWalk() [1/3]

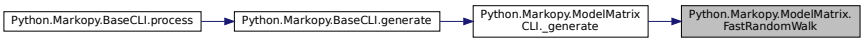
```
def Python.Markopy.ModelMatrix.FastRandomWalk (
            int count,
            str wordlist,
            int minlen,
            int maxlen )  [inherited]
```

Definition at line 48 of file mm.py.

```
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass
```

Referenced by Python.Markopy.ModelMatrixCLI._generate().

Here is the caller graph for this function:



8.30.3.16 FastRandomWalk() [2/3]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [inherited]
```

Random walk on the Matrix-reduced Markov::Model.

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5

This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

n	- Number of passwords to generate.
wordlistFileName	- Filename to write to

Parameters

minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 217 of file modelMatrix.cpp.

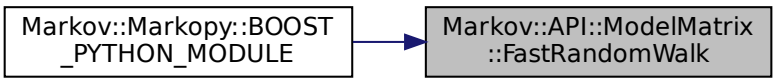
```
00217                                                {
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

References Markov::API::ModelMatrix::FastRandomWalk().

Referenced by Markov::Markopy::BOOST_PYTHON_MODULE().

Here is the call graph for this function:

Here is the caller graph for this function:



8.30.3.17 FastRandomWalk() [3/3]

```
int Markov::API::ModelMatrix::FastRandomWalk (
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20,
            bool bFileIO = true )  [protected], [inherited]
```

Random walk on the Matrix-reduced Markov::Model.

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces Markov::API::MarkovPasswords::Generate runtime by %96.5

This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually replace it.

Parameters

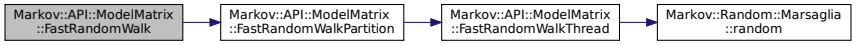
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 204 of file modelMatrix.cpp.

```
00204                                   {
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
      threads);
00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
      threads);
00213     }
00214     return 0;
00215 }
```

References Markov::API::ModelMatrix::FastRandomWalkPartition().
Referenced by Markov::API::ModelMatrix::FastRandomWalk().
Here is the call graph for this function:



Here is the caller graph for this function:

8.30.3.18 FastRandomWalkPartition()

```
void Markov::API::ModelMatrix::FastRandomWalkPartition (
            std::mutex * mlock,
            std::ofstream * wordlist,
            unsigned long int n,
            int minLen,
            int maxLen,
            bool bFileIO,
            int threads )  [protected], [inherited]
```

A single partition of FastRandomWalk event.
Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation

- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

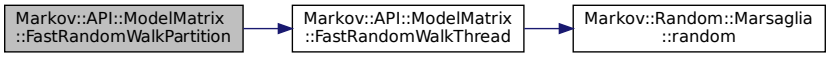
Parameters

mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn
bFileIO	- If false, filename will be ignored and will output to stdout.

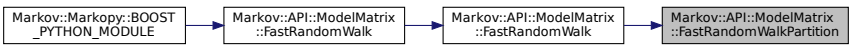
Definition at line 225 of file modelMatrix.cpp.

```
00225                                                                    {
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++){
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
      mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
      wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00239
00240     for(int i=0;i<threads;i++){
00241         threadsV[i]->join();
00242     }
00243 }
```

References Markov::API::ModelMatrix::FastRandomWalkThread().
Referenced by Markov::API::ModelMatrix::FastRandomWalk().
Here is the call graph for this function:



Here is the caller graph for this function:



8.30.3.19 FastRandomWalkThread()

```
void Markov::API::ModelMatrix::FastRandomWalkThread (
            std::mutex * mlock,
```

```
            std::ofstream * wordlist,
            unsigned long int n,
            int minLen,
            int maxLen,
            int id,
            bool bFileIO )  [protected], [inherited]
```

A single thread of a single partition of FastRandomWalk.
A FastRandomWalkPartition will initiate as many of this function as requested.
This function contains the bulk of the generation algorithm.

Parameters

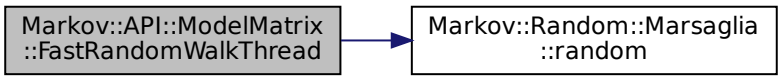
mlock	- mutex lock to distribute to child threads
wordlist	- Reference to the wordlist file to write to
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
id	- **DEPRECATED** Thread id - No longer used
bFileIO	- If false, filename will be ignored and will output to stdout.

Definition at line 153 of file modelMatrix.cpp.

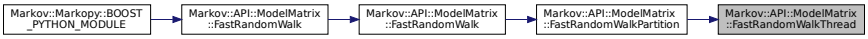
```
00153                                                                   {
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179
00180             if (len >= maxLen)  break;
00181             else if ((next < 0) && (len < minLen)) continue;
00182             else if (next < 0) break;
00183             cur = next;
00184             res[bufferctr + len++] = cur;
00185         }
00186         res[bufferctr + len++] = '\n';
00187         bufferctr+=len;
00188
00189     }
00190     if(bFileIO){
00191         mlock->lock();
00192         *wordlist << res;
00193         mlock->unlock();
00194     }else{
00195         mlock->lock();
00196         std::cout << res;
00197         mlock->unlock();
00198     }
00199     delete res;
00200
00201 }
```

References Markov::API::ModelMatrix::edgeMatrix, Markov::API::ModelMatrix::matrixIndex, Markov::API::ModelMatrix::matrixSize, Markov::Random::Marsaglia::random(), Markov::API::ModelMatrix::totalEdgeWeights, and Markov::API::ModelMatrix::valueMatrix.

Referenced by Markov::API::ModelMatrix::FastRandomWalkPartition().
Here is the call graph for this function:



Here is the caller graph for this function:



8.30.3.20 generate()

```
def Python.Markopy.BaseCLI.generate (
              self,
             str wordlist,
             bool bulk = False )  [inherited]
```

Generate strings from the model.

Parameters

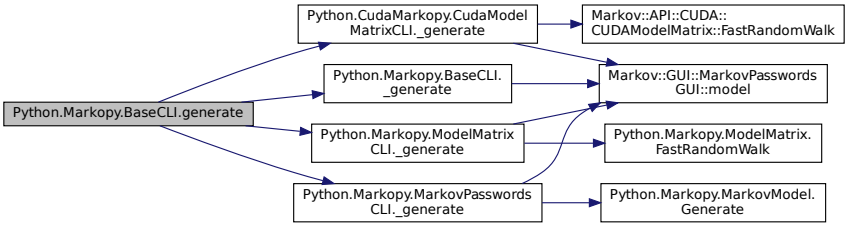
model	model instance
wordlist	wordlist filename
bulk	marks bulk operation with directories

Definition at line 145 of file base.py.

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate(wordlist)
00159 
```

References Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.CudaMarkopy.CudaMarkop
Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.
Referenced by Python.Markopy.BaseCLI.process().

Here is the call graph for this function:



Here is the caller graph for this function:

8.30.3.21 Generate()

```
void Markov::API::MarkovPasswords::Generate (
            unsigned long int n,
            const char * wordlistFileName,
            int minLen = 6,
            int maxLen = 12,
            int threads = 20 )  [inherited]
```

Call Markov::Model::RandomWalk n times, and collect output.
Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See Markov::API::MatrixModel::FastRandomWalk for more information.

Parameters

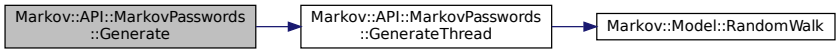
n	- Number of passwords to generate.
wordlistFileName	- Filename to write to
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate
threads	- number of OS threads to spawn

Definition at line 118 of file markovPasswords.cpp.

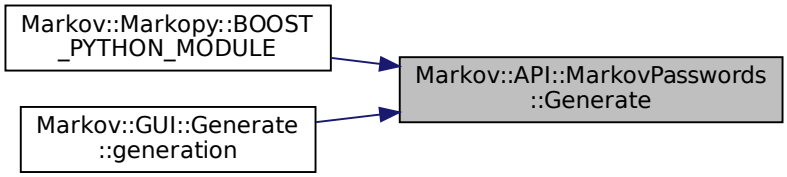
```
00118                                {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
       &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
```

```
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

References Markov::API::MarkovPasswords::GenerateThread().
Referenced by Markov::Markopy::BOOST_PYTHON_MODULE(), and Markov::GUI::Generate::generation().
Here is the call graph for this function:



Here is the caller graph for this function:



8.30.3.22 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
            std::mutex * outputLock,
            unsigned long int n,
            std::ofstream * wordlist,
            int minLen,
            int maxLen )  [private], [inherited]
```

A single thread invoked by the Generate function.
DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Parameters

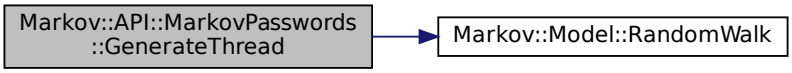
outputLock	- shared mutex lock to lock during output operation. Prevents race condition on write.
n	number of lines to be generated by this thread
wordlist	wordlistfile
minLen	- Minimum password length to generate
maxLen	- Maximum password length to generate

Definition at line 140 of file markovPasswords.cpp.

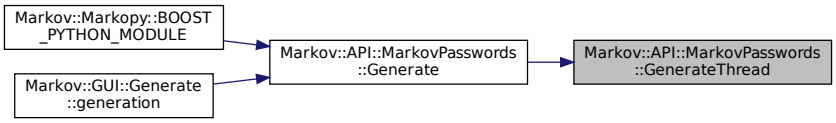
```
00140
                                    {
00141     char* res = new char[maxLen+5];
```

```
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist « res « "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

References Markov::Model< NodeStorageType >::RandomWalk().
Referenced by Markov::API::MarkovPasswords::Generate().
Here is the call graph for this function:



Here is the caller graph for this function:



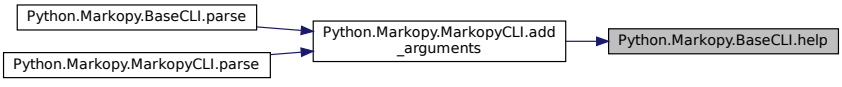
8.30.3.23 help()

```
def Python.Markopy.BaseCLI.help (
              self )  [inherited]
```

Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.
Definition at line 51 of file base.py.

```
00051     def help(self):
00052         "! @brief Handle help strings. Defaults to argparse's help"
00053         self.print_help()
00054
```

References Python.Markopy.BaseCLI.print_help.
Referenced by Python.Markopy.MarkopyCLI.add_arguments().
Here is the caller graph for this function:



8.30.3.24 Import() [1/2]

```
void Markov::API::ModelMatrix::Import (
             const char * filename )  [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.

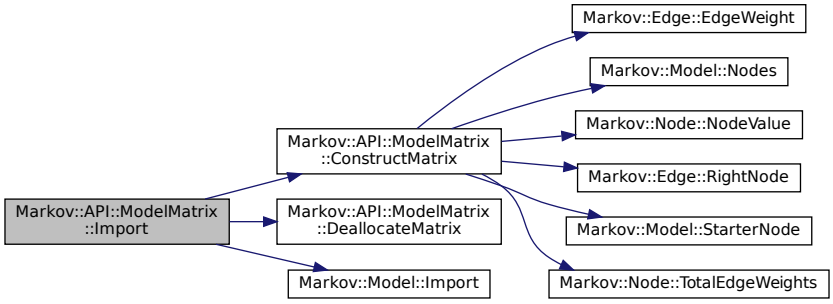
Definition at line 19 of file modelMatrix.cpp.

```
00019                                                   {
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

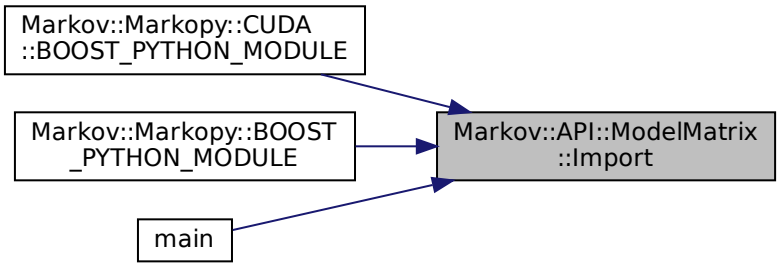
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::Model< NodeStorageType >::Import().

Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), Markov::Markopy::BOOST_PYTHON_MODULE(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



8.30.3.25 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 126 of file model.h.

```
00216                                                   {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout « "cell: " « cell « std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout « oc « "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout « "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout « "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout « int(srcN->NodeValue()) « " --" « e->EdgeWeight() « "--> " «
      int(targetN->NodeValue()) « "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

8.30.3.26 import_model()

```
def Python.Markopy.BaseCLI.import_model (
              self,
             str filename )  [inherited]
```

Import a model file.

Parameters

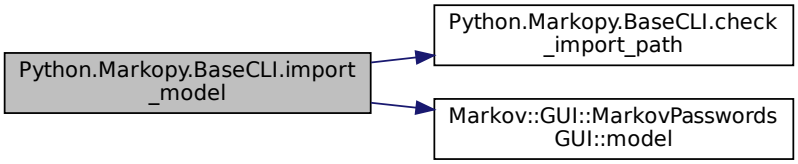
filename	filename to import

Definition at line 77 of file base.py.

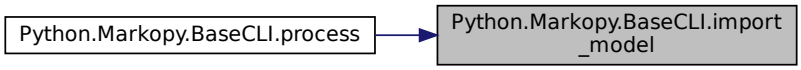
```
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
```

```
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

References Python.Markopy.BaseCLI.check_import_path(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



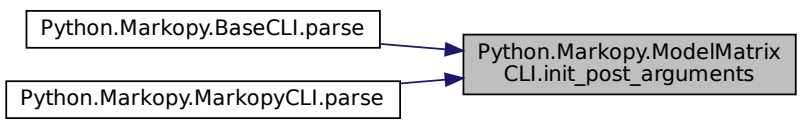
8.30.3.27 init_post_arguments()

```
def Python.Markopy.ModelMatrixCLI.init_post_arguments (
              self )
```

Reimplemented from Python.Markopy.BaseCLI.
Reimplemented in Python.Markopy.MarkopyCLI, Python.CudaMarkopy.CudaModelMatrixCLI, and Python.Markopy.MarkopyCLI.
Definition at line 36 of file mmx.py.

```
00036     def init_post_arguments(self):
00037         super().init_post_arguments()
00038         self.fileIO = not self.args.stdout
00039
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().
Here is the caller graph for this function:



8.30.3.28 Nodes()

```
std::map<char , Node<char >*>* Markov::Model< char >::Nodes ( )  [inline], [inherited]
```

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes;}
```

8.30.3.29 OpenDatasetFile()

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
            const char * filename )  [inherited]
```

Open dataset file and return the ifstream pointer.

Parameters

filename	- Filename to open

Returns

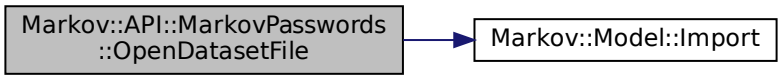
ifstream* to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051                                                                                {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References Markov::Model< NodeStorageType >::Import().

Here is the call graph for this function:



8.30.3.30 OptimizeEdgeOrder()

```
void Markov::Model< char >::OptimizeEdgeOrder  [inherited]
```

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.

```
00265                                                {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout « "Total edges in EdgesV: " « x.second->edgesV.size() « "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
```

```
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout « x.second->edgesV[i]->EdgeWeight() « ", ";
00273         //std::cout « "\n";
00274     }
00275     //std::cout « "Total number of nodes: " « this->nodes.size() « std::endl;
00276     //std::cout « "Total number of edges: " « this->edges.size() « std::endl;
00277 }
```

8.30.3.31 parse()

```
def Python.Markopy.BaseCLI.parse (
            self )  [inherited]
```

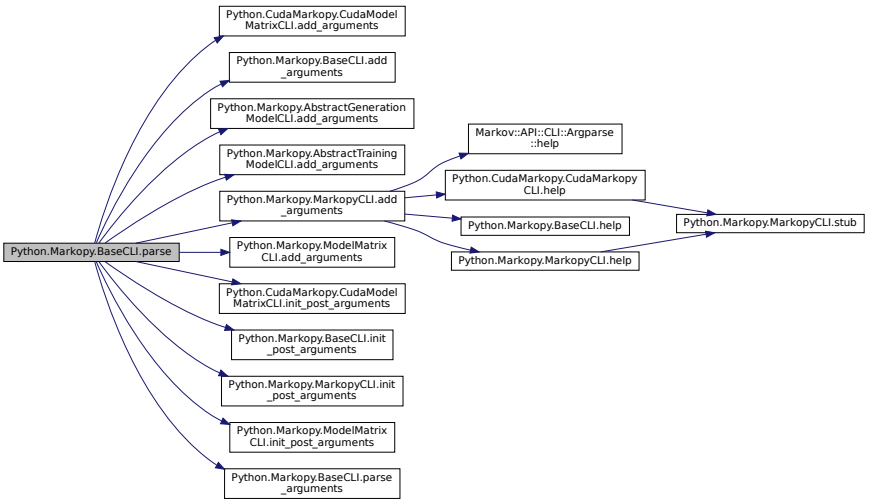
Reimplemented in Python.Markopy.MarkopyCLI, and Python.CudaMarkopy.CudaMarkopyCLI.

Definition at line 55 of file base.py.

```
00055     def parse(self):
00056         "! @brief add, parse and hook arguements"
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMode Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.init_post_arguments(), Python.Markopy.ModelMatrixC and Python.Markopy.BaseCLI.parse_arguments().

Here is the call graph for this function:



8.30.3.32 parse_arguments()

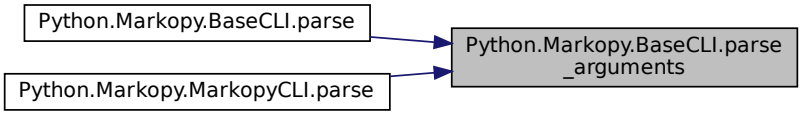
```
def Python.Markopy.BaseCLI.parse_arguments (
            self )  [inherited]
```

Definition at line 73 of file base.py.

```
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by Python.Markopy.BaseCLI.parse(), and Python.Markopy.MarkopyCLI.parse().

Here is the caller graph for this function:



8.30.3.33 process()

```
def Python.Markopy.BaseCLI.process (
              self )  [inherited]
```

Process parameters for operation.

Reimplemented in Python.Markopy.MarkopyCLI.

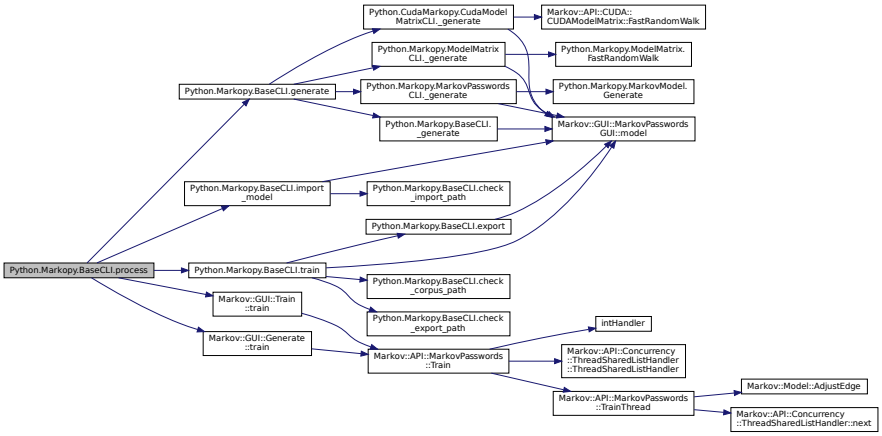
Definition at line 202 of file base.py.

```
00202     def process(self):
00203         """!
00204         @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging.pprint(f"Bulk mode operation chosen.", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
      (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00210                     corpus_list = os.listdir(self.args.dataset)
00211                     for corpus in corpus_list:
00212                         self.import_model(self.args.input)
00213                         logging.pprint(f"Training {self.args.input} with {corpus}", 2)
00214                         output_file_name = corpus
00215                         model_extension = ""
00216                         if "." in self.args.input:
00217                             model_extension = self.args.input.split(".")[-1]
00218                         self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
      f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00219                 else:
00220                     logging.pprint("In bulk training, output and dataset should be a directory.")
00221                     exit(1)
00222
00223             elif (self.args.mode.lower() == "generate"):
00224                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
      (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00225                     model_list = os.listdir(self.args.input)
00226                     print(model_list)
00227                     for input in model_list:
00228                         logging.pprint(f"Generating from {self.args.input}/{input} to
      {self.args.wordlist}/{input}.txt", 2)
00229                         self.import_model(f"{self.args.input}/{input}")
00230                         model_base = input
00231                         if "." in self.args.input:
00232                             model_base = input.split(".")[1]
00233                         self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00234                 else:
00235                     logging.pprint("In bulk generation, input and wordlist should be directory.")
00236
00237         else:
00238             self.import_model(self.args.input)
00239             if (self.args.mode.lower() == "generate"):
00240                 self.generate(self.args.wordlist)
00241
00242
00243             elif (self.args.mode.lower() == "train"):
00244                 self.train(self.args.dataset, self.args.seperator, self.args.output,
      output_forced=True)
00245
00246
00247             elif(self.args.mode.lower() == "combine"):
00248                 self.train(self.args.dataset, self.args.seperator, self.args.output)
00249                 self.generate(self.args.wordlist)
00250
00251
00252             else:
00253                 logging.pprint("Invalid mode arguement given.")
```

```
00254                 logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255                 exit(5)
00256
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.generate(), Python.Markopy.BaseCLI.import_model(), Markov::GUI::Generate.train(), Markov::GUI::Train.train(), and Python.Markopy.BaseCLI.train().

Here is the call graph for this function:



8.30.3.34 RandomWalk()

```
char * Markov::Model< char >::RandomWalk (
            Markov::Random::RandomEngine * randomEngine,
            int minSetting,
            int maxSetting,
            NodeStorageType * buffer )  [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criateria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes Markov::Random::RandomEngine as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so Markov::Random::Marsaglia is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout « res « "\n";
 }
```

Parameters

randomEngine	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
minSetting	Minimum number of characters to generate
maxSetting	Maximum number of character to generate
buffer	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307                                                       {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue();
00327     }
00328
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

8.30.3.35 Save()

```
std::ofstream * Markov::API::MarkovPasswords::Save (
            const char * filename )  [inherited]
```

Export model to file.

Parameters

filename	- Export filename.

Returns

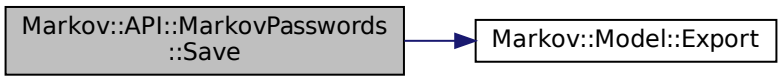
std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```
00106                                                                   {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References Markov::Model< NodeStorageType >::Export().

Here is the call graph for this function:



8.30.3.36 StarterNode()

```
Node<char >* Markov::Model< char >::StarterNode ( )  [inline], [inherited]
```

Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file model.h.

```
00171 { return starterNode;}
```

8.30.3.37 Train()

```
void Markov::API::ModelMatrix::Train (
            const char * datasetFileName,
            char delimiter,
            int threads )  [inherited]
```

Train the model with the dataset file.

Parameters

datasetFileName	- Ifstream* to the dataset. If null, use class member
delimiter	- a character, same as the delimiter in dataset content
threads	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

Construct the matrix when done.

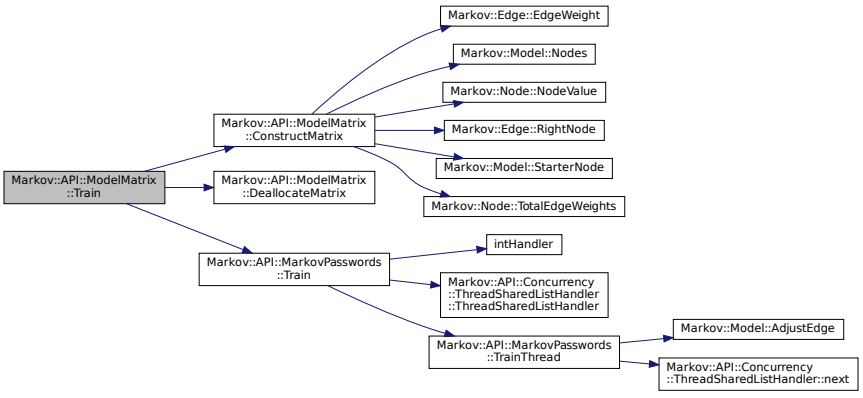
Definition at line 25 of file modelMatrix.cpp.

```
00025                                                                                                {
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

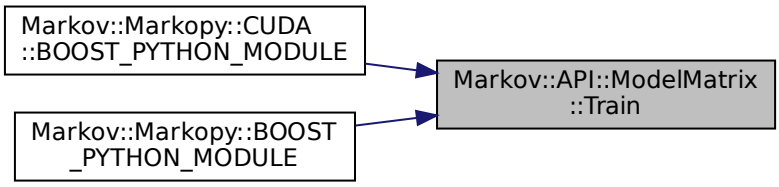
References Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::MarkovPasswords::Train().

Referenced by Markov::Markopy::CUDA::BOOST_PYTHON_MODULE(), and Markov::Markopy::BOOST_PYTHON_MODULE().

Here is the call graph for this function:



Here is the caller graph for this function:



8.30.3.38 train()

```
def Python.Markopy.BaseCLI.train (
               self,
              str dataset,
              str seperator,
              str output,
              bool output_forced = False,
              bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

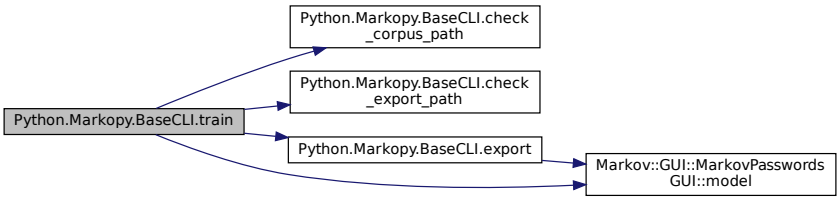
model	Model instance
dataset	filename for the dataset
seperator	seperator used with the dataset
output	output filename
output_forced	force overwrite
bulk	marks bulk operation with directories

Definition at line 94 of file base.py.

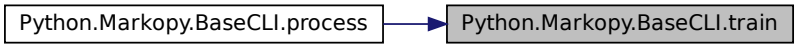
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105 
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else""} and -s/--seperator parameters. Exiting.")
00108             return False
00109 
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113 
00114         if not self.check_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117 
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121 
```

```
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125 
00126         logging.pprint(f'Starting training.', 3)
00127         self.model.Train(dataset,seperator, int(self.args.threads))
00128         logging.pprint(f'Training completed.', 2)
00129 
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.export(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135 
00136         return True
00137 
```

References Python.CudaMarkopy.CudaMarkopyCLI.args, Python.Markopy.BaseCLI.args, Python.Markopy.MarkopyCLI.args, Python.Markopy.BaseCLI.check_corpus_path(), Python.Markopy.BaseCLI.check_export_path(), Python.Markopy.BaseCLI.export(), Python.CudaMarkopy.CudaModelMatrixCLI.model, Python.Markopy.BaseCLI.model, Python.Markopy.ModelMatrixCLI.model, Python.Markopy.MarkovPasswordsCLI.model, and Markov::GUI::MarkovPasswordsGUI.model().
Referenced by Python.Markopy.BaseCLI.process().
Here is the call graph for this function:



Here is the caller graph for this function:



8.30.3.39 TrainThread()

```
void Markov::API::MarkovPasswords::TrainThread (
            Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
            char delimiter )  [private], [inherited]
```

A single thread invoked by the Train function.

Parameters

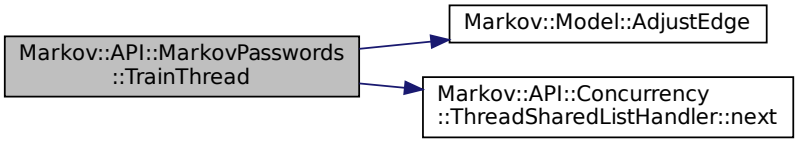
listhandler	- Listhandler class to read corpus from
delimiter	- a character, same as the delimiter in dataset content

Definition at line 85 of file markovPasswords.cpp.

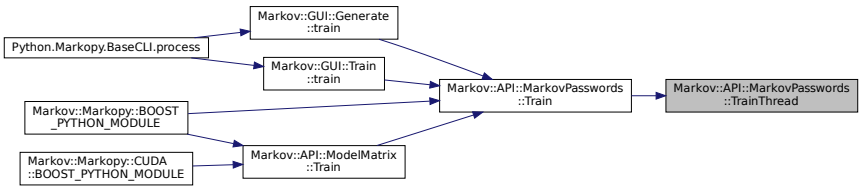
```
00085 
                {
00086     char format_str[] ="%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
```

```
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
       "%ld,%s"
00097 #else
00098         sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100         this->AdjustEdge((const char*)linebuf, oc);
00101         delete linebuf;
00102     }
00103 }
```

References Markov::Model< NodeStorageType >::AdjustEdge(), keepRunning, and Markov::API::Concurrency::ThreadSharedListHa
Referenced by Markov::API::MarkovPasswords::Train().
Here is the call graph for this function:



Here is the caller graph for this function:



8.30.4 Member Data Documentation

8.30.4.1 args

`Python.Markopy.BaseCLI.args [inherited]`
Definition at line 75 of file base.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.generate Python.Markopy.MarkopyCLI.help(), Python.Markopy.BaseCLI.init_post_arguments(), Python.Markopy.MarkopyCLI.parse(), Python.CudaMarkopy.CudaMarkopyCLI.parse_fail(), Python.Markopy.BaseCLI.process(), and Python.Markopy.BaseCLI.train().

8.30.4.2 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]`
Definition at line 123 of file markovPasswords.h.

8.30.4.3 edgeMatrix

`char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]`
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line 175 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.30.4.4 edges

`std::vector<Edge<char >*> Markov::Model< char >::edges [private], [inherited]`
A list of all edges in this model.
Definition at line 204 of file model.h.

8.30.4.5 fileIO

`Python.Markopy.ModelMatrixCLI.fileIO`
Definition at line 38 of file mmx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), and Python.Markopy.ModelMatrixCLI._generate().

8.30.4.6 matrixIndex

`char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]`
to hold the Matrix index (To hold the orders of 2-D arrays')
Definition at line 190 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.30.4.7 matrixSize

`int Markov::API::ModelMatrix::matrixSize [protected], [inherited]`
to hold Matrix size
Definition at line 185 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), Markov::API::ModelMatrix::FastRandomWalkThread(), and Markov::API::CUDA::CUDAMoc

8.30.4.8 model

`Python.Markopy.ModelMatrixCLI.model`
Definition at line 30 of file mmx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI._generate(), Python.Markopy.BaseCLI._generate(), Python.Markopy.ModelMatrixCLI._generate(), Python.Markopy.MarkovPasswordsCLI._generate(), Python.Markopy.BaseCLI.export(), Python.Markopy.BaseCLI.import_model(), and Python.Markopy.BaseCLI.train().

8.30.4.9 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]`
Dataset file input of our system

Definition at line 124 of file markovPasswords.h.

8.30.4.10 nodes

std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.

8.30.4.11 outputFile

std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]
File to save model of our system

Definition at line 125 of file markovPasswords.h.

8.30.4.12 parser

Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.__init__(), Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments(), Python.Markopy.BaseCLI.add_arguments(), Python.Markopy.AbstractGenerationModelCLI.add_arguments(), Python.Markopy.AbstractTrainingModelCLI.add_arguments(), Python.Markopy.MarkopyCLI.add_arguments(), Python.Markopy.ModelMatrixCLI.add_arguments(), Python.CudaMarkopy.CudaMarkopyCLI.help(), and Python.Markopy.MarkopyCLI.

8.30.4.13 print_help

Python.Markopy.BaseCLI.print_help [inherited]
Definition at line 39 of file base.py.
Referenced by Python.Markopy.BaseCLI.help(), and Python.Markopy.MarkopyCLI.help().

8.30.4.14 ready

bool Markov::API::ModelMatrix::ready [protected], [inherited]
True when matrix is constructed. False if not.
Definition at line 200 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::ModelMatrix().

8.30.4.15 starterNode

Node<char >* Markov::Model< char >::starterNode [private], [inherited]
Starter Node of this model.
Definition at line 198 of file model.h.

8.30.4.16 totalEdgeWeights

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]
Array of the Total Edge Weights.
Definition at line 195 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), and Markov::API::ModelMatrix::FastRandomWalkThread().

8.30.4.17 valueMatrix

`long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]`

2-d Integer array for the value Matrix (For the weights of Edges)

Definition at line 180 of file modelMatrix.h.

Referenced by Markov::API::ModelMatrix::ConstructMatrix(), Markov::API::ModelMatrix::DeallocateMatrix(), Markov::API::ModelMatrix::DumpJSON(), and Markov::API::ModelMatrix::FastRandomWalkThread().

The documentation for this class was generated from the following file:

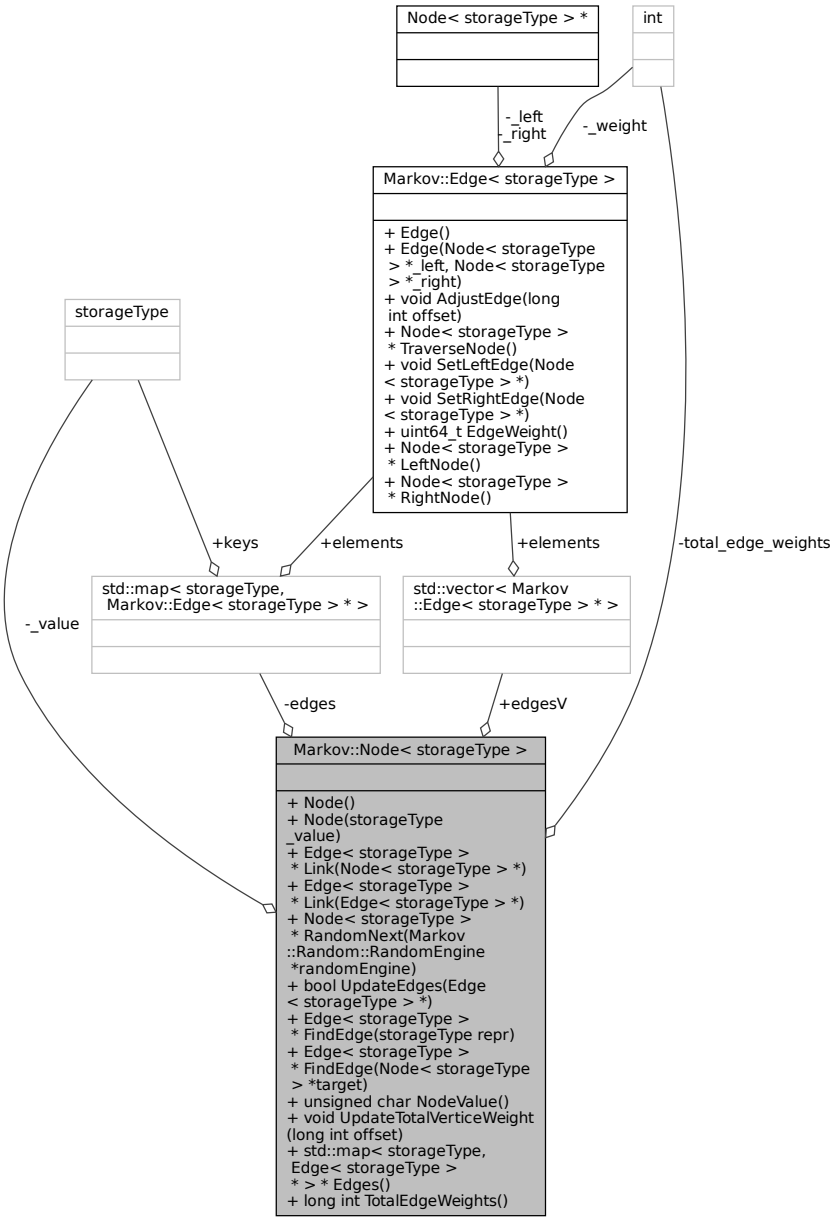
- Markopy/Markopy/src/CLI/mmx.py

8.31 Markov::Node< storageType > Class Template Reference

A node class that for the vertices of model. Connected with eachother using Edge.

`#include <node.h>`

Collaboration diagram for Markov::Node< storageType >:



Public Member Functions

- Node ()

  Default constructor. Creates an empty Node.
- Node (storageType _value)

  Constructor. Creates a Node with no edges and with given NodeValue.
- Edge< storageType > * Link (Node< storageType > *)

  Link this node with another, with this node as its source.
- Edge< storageType > * Link (Edge< storageType > *)

  Link this node with another, with this node as its source.
- Node< storageType > * RandomNext (Markov::Random::RandomEngine *randomEngine)

  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

  Character representation of this node. 0 for starter, 0xff for terminator.
- long int total_edge_weights

  Total weights of the vertices, required by RandomNext.
- std::map< storageType, Edge< storageType > * > edges

  A map of all edges connected to this node, where this node is at the LeftNode. Map is indexed by unsigned char, which is the character representation of the node.

8.31.1 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 24 of file node.h.

8.31.2 Constructor & Destructor Documentation

8.31.2.1 Node() [1/2]

```
template<typename storageType >
Markov::Node< storageType >::Node
```

Default constructor. Creates an empty Node.

Definition at line 209 of file node.h.

```
00209                                {
00210     this->_value = 0;
00211     this->total_edge_weights = 0L;
00212 };
```

8.31.2.2 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 203 of file node.h.

```
00203                                                {
00204     this->_value = _value;
00205     this->total_edge_weights = 0L;
00206 };
```

8.31.3 Member Function Documentation

8.31.3.1 Edges()

```
template<typename storageType >
std::map< storageType, Markov::Edge< storageType > * > * Markov::Node< storageType >::Edges
[inline]
```

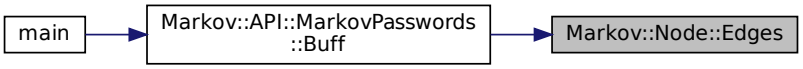
return edges

Definition at line 272 of file node.h.

```
00272                                                                                {
00273     return &(this->edges);
00274 }
```

Referenced by Markov::API::MarkovPasswords::Buff().

Here is the caller graph for this function:



8.31.3.2 FindEdge() [1/2]

```
template<typename storageType >
Edge<storageType>* Markov::Node< storageType >::FindEdge (
            Node< storageType > * target )
```

Find an edge with its pointer. Avoid unless neccessary because comptutational cost of find by character is cheaper (because of std::map)

Parameters

target	- target node.

Returns

Edge that is connected between this node, and the target node.

8.31.3.3 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 260 of file node.h.

```
00260                                                                    {
00261     auto e = this->edges.find(repr);
00262     if (e == this->edges.end()) return NULL;
00263     return e->second;
00264 };
```

8.31.3.4 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 227 of file node.h.

```
00227                                                                    {
00228     v->SetLeftEdge(this);
00229     this->UpdateEdges(v);
00230     return v;
00231 }
```

8.31.3.5 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 220 of file node.h.

```
00220                                                                    {
00221     Markov::Edge<storageType>* v = new Markov::Edge<storageType>(this, n);
00222     this->UpdateEdges(v);
00223     return v;
00224 }
```

8.31.3.6 NodeValue()

```
template<typename storageType >
unsigned char Markov::Node< storageType >::NodeValue  [inline]
```

Return character representation of this node.

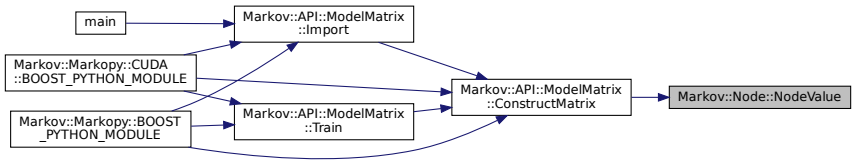
Returns

character representation at _value.

Definition at line 215 of file node.h.

```
00215                                                {
00216     return _value;
00217 }
```

References Markov::Node< storageType >::_value.
Referenced by Markov::API::ModelMatrix::ConstructMatrix().
Here is the caller graph for this function:



8.31.3.7 RandomNext()

```
template<typename storageType >
Markov::Node< storageType > * Markov::Node< storageType >::RandomNext (
            Markov::Random::RandomEngine * randomEngine )
```

Chose a random node from the list of edges, with regards to its EdgeWeight, and TraverseNode to that.
This operation is done by generating a random number in range of 0-this.total_edge_weights, and then iterating over the list of edges. At each step, EdgeWeight of the edge is subtracted from the random number, and once it is 0, next node is selected.

Returns

Node that was chosen at EdgeWeight biased random.

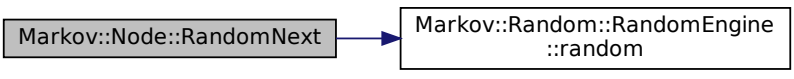
Example Use: Use randomNext to do a random walk on the model

```
char* buffer[64];
Markov::Model<char> model;
model.Import("model.mdl");
   Markov::Node<char>* n = model.starterNode;
int len = 0;
Markov::Node<char>* temp_node;
while (true) {
   temp_node = n->RandomNext(randomEngine);
   if (len >= maxSetting) {
       break;
   }
   else if ((temp_node == NULL) && (len < minSetting)) {
       continue;
   }
   else if (temp_node == NULL){
       break;
   }

   n = temp_node;
   buffer[len++] = n->NodeValue();
}
```

Definition at line 234 of file node.h.

```
00234                                                                                                                {
00235
00236     //get a random NodeValue in range of total_vertice_weight
00237     long int selection = randomEngine->random() %
      this->total_edge_weights;//distribution()(generator());// distribution(generator);
00238     //make absolute, no negative modulus values wanted
00239     //selection = (selection >= 0) ? selection : (selection + this->total_edge_weights);
00240     for(int i=0;i<this->edgesV.size();i++){
00241         selection -= this->edgesV[i]->EdgeWeight();
00242         if (selection < 0) return this->edgesV[i]->TraverseNode();
00243     }
00244
00245     //if this assertion is reached, it means there is an implementation error above
00246     std::cout « "This should never be reached (node failed to walk to next)\n"; //cant assert from
      child thread
00247     assert(true && "This should never be reached (node failed to walk to next)");
00248     return NULL;
00249 }
```

References Markov::Random::RandomEngine::random().
Here is the call graph for this function:



8.31.3.8 TotalEdgeWeights()

```
template<typename storageType >
long int Markov::Node< storageType >::TotalEdgeWeights  [inline]
```

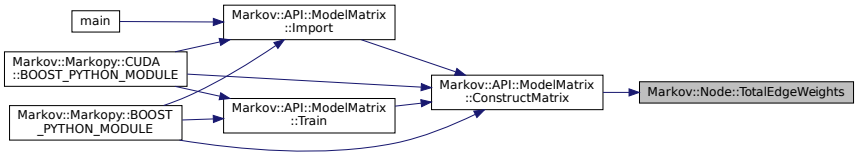
return total edge weights

Definition at line 277 of file node.h.

```
00277                                                                    {
00278     return this->total_edge_weights;
00279 }
```

Referenced by Markov::API::ModelMatrix::ConstructMatrix().

Here is the caller graph for this function:



8.31.3.9 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 252 of file node.h.

```
00252                                                                    {
00253     this->edges.insert({ v->RightNode()->NodeValue(), v });
00254     this->edgesV.push_back(v);
00255     //this->total_edge_weights += v->EdgeWeight();
00256     return v->TraverseNode();
00257 }
```

8.31.3.10 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 267 of file node.h.

```
00267                                                                    {
00268     this->total_edge_weights += offset;
00269 }
```

8.31.4 Member Data Documentation

8.31.4.1 _value

```
template<typename storageType >
storageType Markov::Node< storageType >::_value  [private]
```

Character representation of this node. 0 for starter, 0xff for terminator.
Definition at line 179 of file node.h.
Referenced by Markov::Node< storageType >::NodeValue().

8.31.4.2 edges

```
template<typename storageType >
std::map<storageType, Edge<storageType>*> Markov::Node< storageType >::edges  [private]
```

A map of all edges connected to this node, where this node is at the LeftNode. Map is indexed by unsigned char, which is the character representation of the node.
Definition at line 190 of file node.h.

8.31.4.3 edgesV

```
template<typename storageType >
std::vector<Edge<storageType>*> Markov::Node< storageType >::edgesV
```

Definition at line 173 of file node.h.

8.31.4.4 total_edge_weights

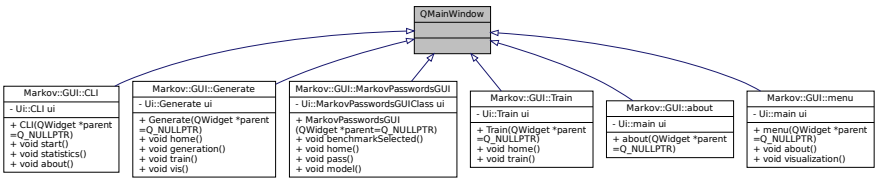
```
template<typename storageType >
long int Markov::Node< storageType >::total_edge_weights  [private]
```

Total weights of the vertices, required by RandomNext.
Definition at line 184 of file node.h.
The documentation for this class was generated from the following files:

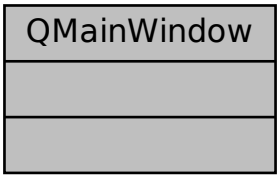
- Markopy/MarkovModel/src/model.h
- Markopy/MarkovModel/src/node.h

8.32 QMainWindow Class Reference

Inheritance diagram for QMainWindow:



Collaboration diagram for QMainWindow:



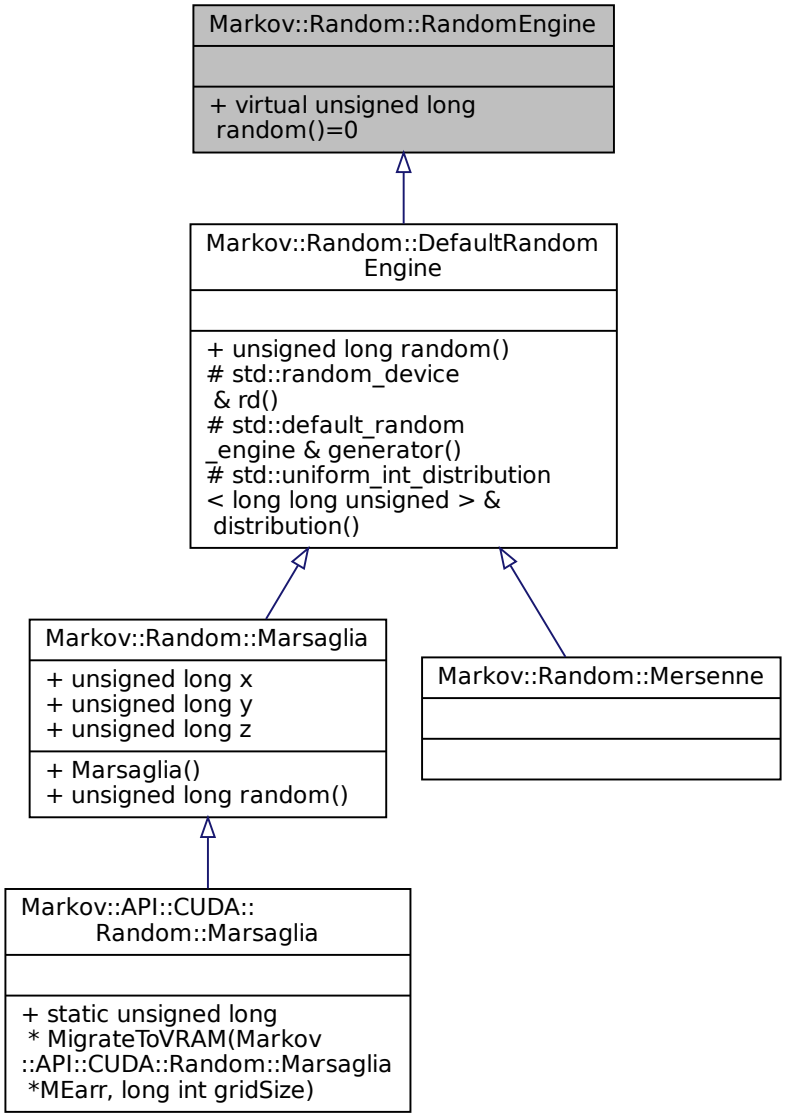
The documentation for this class was generated from the following file:

- Markopy/MarkovPasswordsGUI/src/CLI.h

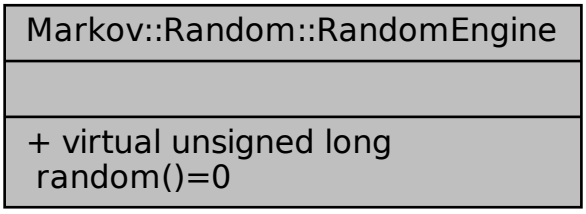
8.33 Markov::Random::RandomEngine Class Reference

An abstract class for Random Engine.
`#include <random.h>`

Inheritance diagram for Markov::Random::RandomEngine:



Collaboration diagram for Markov::Random::RandomEngine:



Public Member Functions

- virtual unsigned long random ()=0

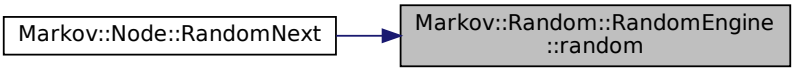
8.33.1 Detailed Description

An abstract class for Random Engine.
This class is used for generating random numbers, which are used for random walking on the graph.
Main reason behind allowing different random engines is that some use cases may favor performance, while some favor good random.
Mersenne can be used for truer random, while Marsaglia can be used for deterministic but fast random.
Definition at line 30 of file random.h.

8.33.2 Member Function Documentation

8.33.2.1 random()

`virtual unsigned long Markov::Random::RandomEngine::random ( ) [inline], [pure virtual]`
Implemented in Markov::Random::Marsaglia, and Markov::Random::DefaultRandomEngine.
Referenced by Markov::Node< storageType >::RandomNext().
Here is the caller graph for this function:



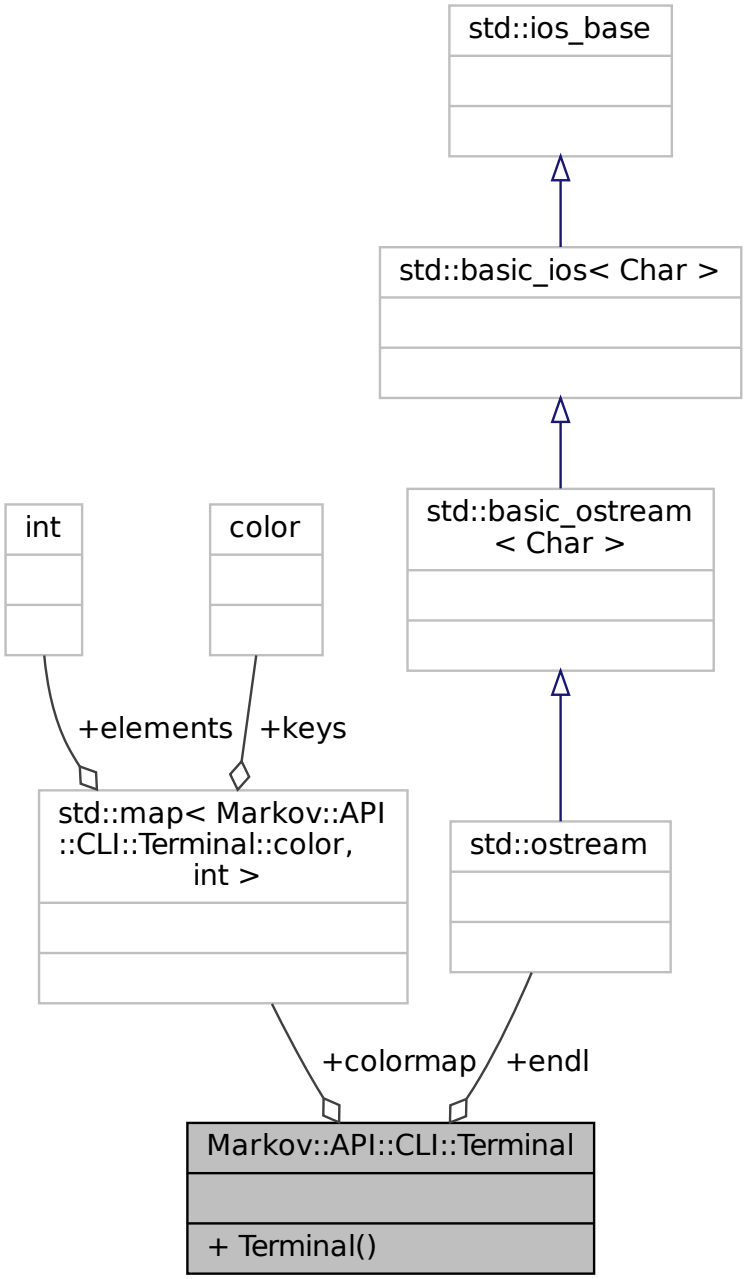
The documentation for this class was generated from the following file:

- Markopy/MarkovModel/src/random.h

8.34 Markov::API::CLI::Terminal Class Reference

pretty colors for Terminal. Windows Only.
`#include <term.h>`

Collaboration diagram for Markov::API::CLI::Terminal:



Public Types

- enum color {
  RESET , BLACK , RED , GREEN ,
  YELLOW , BLUE , MAGENTA , CYAN ,
  WHITE , LIGHTGRAY , DARKGRAY , BROWN }

Public Member Functions

- Terminal ()

Static Public Attributes

- static std::map< Markov::API::CLI::Terminal::color, int > colormap
- static std::ostream endl

8.34.1 Detailed Description

pretty colors for Terminal. Windows Only.
Definition at line 25 of file term.h.

8.34.2 Member Enumeration Documentation

8.34.2.1 color

```
enum Markov::API::CLI::Terminal::color
```

Enumerator

RESET	
BLACK	
RED	
GREEN	
YELLOW	
BLUE	
MAGENTA	
CYAN	
WHITE	
LIGHTGRAY	
DARKGRAY	
BROWN	

Definition at line 33 of file term.h.

```
00033 { RESET, BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, LIGHTGRAY, DARKGRAY, BROWN };
```

8.34.3 Constructor & Destructor Documentation

8.34.3.1 Terminal()

```
Terminal::Terminal ( )
```

Default constructor. Get references to stdout and stderr handles.
Definition at line 62 of file term.cpp.

```
00062                   {
00063     /*this->;*/
00064 }
```

8.34.4 Member Data Documentation

8.34.4.1 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 39 of file term.h.
Referenced by operator<<().

8.34.4.2 endl

```
std::ostream Markov::API::CLI::Terminal::endl  [static]
```

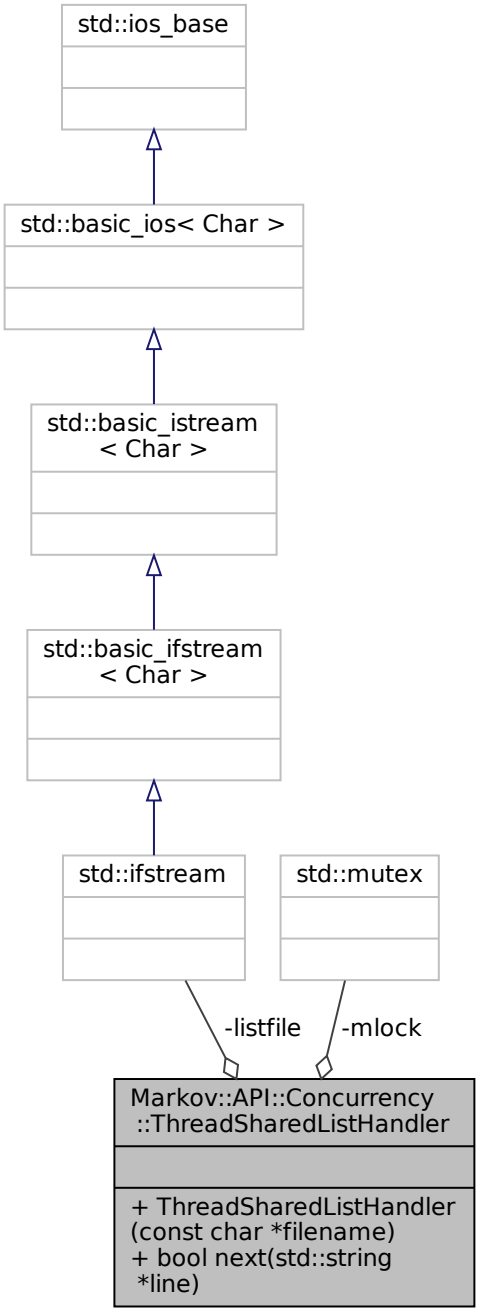
Definition at line 44 of file term.h.
The documentation for this class was generated from the following files:

- Markopy/MarkovAPICLI/src/color/term.h
- Markopy/MarkovAPICLI/src/color/term.cpp

8.35 Markov::API::Concurrency::ThreadSharedListHandler Class Reference

Simple class for managing shared access to file.
`#include <threadSharedListHandler.h>`

Collaboration diagram for Markov::API::Concurrency::ThreadSharedListHandler:



Public Member Functions

- ThreadSharedListHandler (const char ∗filename)

  Construct the Thread Handler with a filename.

- bool next (std::string ∗line)

  Read the next line from the file.

Private Attributes

- std::ifstream listfile
- std::mutex mlock

8.35.1 Detailed Description

Simple class for managing shared access to file.
This class maintains the handover of each line from a file to multiple threads.
When two different threads try to read from the same file while reading a line isn't completed, it can have unexpected results. Line might be split, or might be read twice. This class locks the read action on the list until a line is completed, and then proceeds with the handover.
Definition at line 25 of file threadSharedListHandler.h.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 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

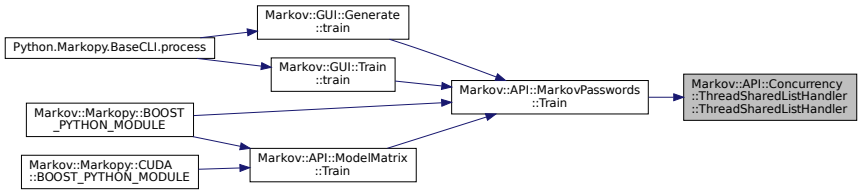
filename	Filename for the file to manage.

Definition at line 12 of file threadSharedListHandler.cpp.

```
00012                                                                                    {
00013     this->listfile;
00014     this->listfile.open(filename, std::ios_base::binary);
00015 }
```

References listfile.

Referenced by Markov::API::MarkovPasswords::Train().
Here is the caller graph for this function:



8.35.3 Member Function Documentation

8.35.3.1 next()

```
bool Markov::API::Concurrency::ThreadSharedListHandler::next (
            std::string * line )
```

Read the next line from the file.
This action will be blocked until another thread (if any) completes the read operation on the file.
Example Use: Simple file read

```
ThreadSharedListHandler listhandler("test.txt");
std::string line;
std::cout « listhandler->next(&line) « "\n";
```

Definition at line 18 of file threadSharedListHandler.cpp.

```
00018                                                                   {
00019     bool res = false;
00020     this->mlock.lock();
00021     res = (std::getline(this->listfile,*line,'\n'))? true : false;
00022     this->mlock.unlock();
00023
00024     return res;
00025 }
```

References listfile, and mlock.
Referenced by Markov::API::MarkovPasswords::TrainThread().
Here is the caller graph for this function:

8.35.4 Member Data Documentation

8.35.4.1 listfile

```
std::ifstream Markov::API::Concurrency::ThreadSharedListHandler::listfile  [private]
```

Definition at line 95 of file threadSharedListHandler.h.
Referenced by next(), and ThreadSharedListHandler().

8.35.4.2 mlock

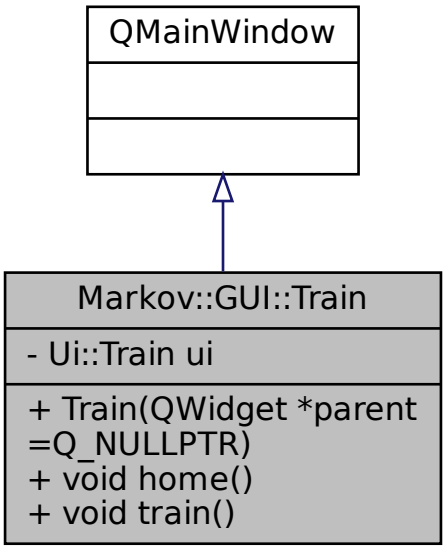
```
std::mutex Markov::API::Concurrency::ThreadSharedListHandler::mlock  [private]
```

Definition at line 96 of file threadSharedListHandler.h.
Referenced by next().
The documentation for this class was generated from the following files:

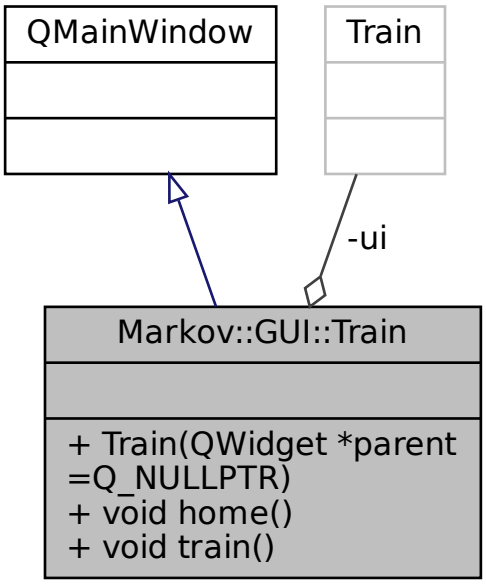
- Markopy/MarkovAPI/src/threadSharedListHandler.h
- Markopy/MarkovAPI/src/threadSharedListHandler.cpp

8.36 Markov::GUI::Train Class Reference

QT Training page class.
`#include <Train.h>`
Inheritance diagram for Markov::GUI::Train:



Collaboration diagram for Markov::GUI::Train:



Public Slots

- void home ()
- void train ()

Public Member Functions

- Train (QWidget ∗parent=Q_NULLPTR)

Private Attributes

- Ui::Train ui

8.36.1 Detailed Description

QT Training page class.
Definition at line 15 of file Train.h.

8.36.2 Constructor & Destructor Documentation

8.36.2.1 Train()

```
Markov::GUI::Train::Train (
            QWidget * parent = Q_NULLPTR )
```

Definition at line 22 of file Train.cpp.

```
00023     : QMainWindow(parent)
00024 {
00025     ui.setupUi(this);
00026
```

```
00027
00028
00029     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {train(); });
00030     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {home(); });
00031     //QObject::connect(ui.pushButton_3, &QPushButton::clicked, this, [this] {goGenerate(); });
00032
00033     //ui.pushButton_3->setVisible(false);
00034
00035
00036 }
```

References ui.

8.36.3 Member Function Documentation

8.36.3.1 home

```
void Markov::GUI::Train::home ( )  [slot]
```

Definition at line 73 of file Train.cpp.

```
00073                          {
00074      CLI* w = new CLI;
00075      w->show();
00076      this->close();
00077 }
```

8.36.3.2 train

```
void Markov::GUI::Train::train ( )  [slot]
```

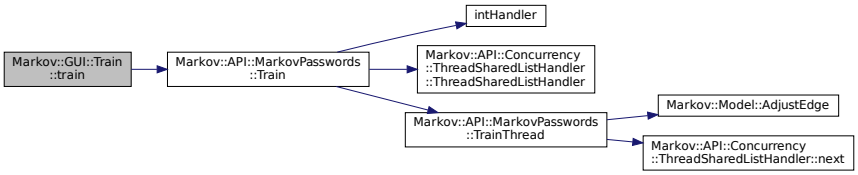
Definition at line 38 of file Train.cpp.

```
00038                           {
00039
00040
00041
00042     QString file_name = QFileDialog::getOpenFileName(this, "Open a File", QDir::homePath());
00043     QFile file(file_name);
00044
00045     if (!file.open(QFile::ReadOnly | QFile::Text)) {
00046         QMessageBox::warning(this, "Error", "File Not Open!");
00047     }
00048     QTextStream in(&file);
00049     QString text = in.readAll();
00050     ui.plainTextEdit->setPlainText(text);
00051
00052
00053     char* cstr;
00054     std::string fname = file_name.toStdString();
00055     cstr = new char[fname.size() + 1];
00056     strcpy(cstr, fname.c_str());
00057
00058
00059
00060     char a=',';
00061     Markov::API::MarkovPasswords mp;
00062     mp.Import("models/2gram.mdl");
00063     mp.Train(cstr, a, 10); //please parameterize this hardcoded 10 threads
00064     mp.Export("models/finished.mdl");
00065
00066     ui.label_2->setText("Training DONE!");
00067     //ui.pushButton_3->setVisible(true);
00068
00069
00070     file.close();
00071 }
```

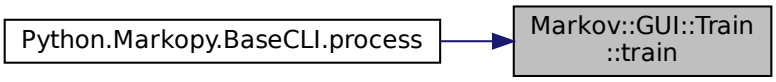
References Markov::API::MarkovPasswords::Train(), and ui.

Referenced by Python.Markopy.BaseCLI::process().

Here is the call graph for this function:



Here is the caller graph for this function:



8.36.4 Member Data Documentation

8.36.4.1 ui

`Ui::Train Markov::GUI::Train::ui [private]`

Definition at line 21 of file Train.h.

Referenced by train(), and Train().

The documentation for this class was generated from the following files:

- Markopy/MarkovPasswordsGUI/src/Train.h
- Markopy/MarkovPasswordsGUI/src/Train.cpp

CHAPTER9

File Documentation

9.1 Markopy/CudaMarkopy/src/CLI/cudamarkopy.py File Reference

base command line interface for python

Classes

- class Python.CudaMarkopy.CudaMarkopyCLI

  CUDA extension to MarkopyCLI.

Namespaces

- cudamarkopy
- Python.CudaMarkopy

  wrapper scripts for CudaMarkopy

Variables

- cudamarkopy.spec = spec_from_loader("markopy", SourceFileLoader("markopy", os.path.abspath("markopy.↩ py")))
- cudamarkopy.markopy = module_from_spec(spec)
- cudamarkopy.mp = CudaMarkopyCLI()

9.1.1 Detailed Description

base command line interface for python
Definition in file cudamarkopy.py.

9.2 cudamarkopy.py

```
00001 #!/usr/bin/python3
00002
00003
00007
00008
00012
00013 import sys
00014 import os
00015 from importlib.util import spec_from_loader, module_from_spec
00016 from importlib.machinery import SourceFileLoader, ExtensionFileLoader
00017 import inspect
00018 try:
00019     spec = spec_from_loader("markopy", SourceFileLoader("markopy", os.path.abspath("markopy.py")))
00020     markopy = module_from_spec(spec)
00021     spec.loader.exec_module(markopy)
00022 except (ModuleNotFoundError,FileNotFoundError) as e:
00023     if(os.path.exists("../../../Markopy/src/CLI/markopy.py")):
00024         spec = spec_from_loader("markopy", SourceFileLoader("markopy",
       "../../../Markopy/src/CLI/markopy.py"))
00025         markopy = module_from_spec(spec)
00026         spec.loader.exec_module(markopy)
00027
00028 try:
00029     from cudammx import CudaModelMatrixCLI
00030     from mmx import ModelMatrixCLI
00031     from mp import MarkovPasswordsCLI
00032
00033 except ModuleNotFoundError as e:
00034     #print("Working in development mode. Trying to load markopy.py from ../../../Markopy/")
00035     if(os.path.exists("../../../Markopy/src/CLI/cudammx.py")):
```

```
00036         sys.path.insert(1, '../../../Markopy/src/CLI/')
00037         from cudammx import CudaModelMatrixCLI
00038         from mmx import ModelMatrixCLI
00039         from mp import MarkovPasswordsCLI
00040     else:
00041         raise e
00042
00043 from termcolor import colored
00044
00045
00046 class CudaMarkopyCLI(markopy.MarkopyCLI, CudaModelMatrixCLI):
00047     """!
00048          @brief CUDA extension to MarkopyCLI. Adds CudaModelMatrixCLI to the interface.
00049          @belongsto Python::CudaMarkopy
00050          @extends Python::Markopy::MarkopyCLI
00051          @extends Python::CudaMarkopy::CudaModelMatrixCLI
00052     """
00053     def __init__(self) -> None:
00054         "! @brief initialize CLI selector"
00055         markopy.MarkopyCLI.__init__(self,add_help=False)
00056         self.parserparser.epilog+=f"""
00057         {__file__.split("/")[-1]} -mt CUDA generate trained.mdl -n 500 -w output.txt
00058             Import trained.mdl, and generate 500 lines to output.txt
00059         """
00060
00061     def help(self):
00062         "! @brief overload help string"
00063         self.parserparser.print_help = self.stubstub
00064         self.argsargsargsargs = self.parserparser.parse_known_args()[0]
00065         if(self.argsargsargsargs.model_type!="_MMX"):
00066             if(self.argsargsargsargs.model_type=="MP"):
00067                 mp = MarkovPasswordsCLI()
00068                 mp.add_arguments()
00069                 mp.parser.print_help()
00070             elif(self.argsargsargsargs.model_type=="MMX"):
00071                 mp = ModelMatrixCLI()
00072                 mp.add_arguments()
00073                 mp.parser.print_help()
00074             elif(self.argsargsargsargs.model_type == "CUDA"):
00075                 mp = CudaModelMatrixCLI()
00076                 mp.add_arguments()
00077                 mp.parser.print_help()
00078         else:
00079             print(colored("Model Mode selection choices:", "green"))
00080             self.print_helpprint_help()
00081             print(colored("Following are applicable for -mt MP mode:", "green"))
00082             mp = MarkovPasswordsCLI()
00083             mp.add_arguments()
00084             mp.parser.print_help()
00085             print(colored("Following are applicable for -mt MMX mode:", "green"))
00086             mp = ModelMatrixCLI()
00087             mp.add_arguments()
00088             mp.parser.print_help()
00089             print(colored("Following are applicable for -mt CUDA mode:", "green"))
00090             mp = CudaModelMatrixCLI()
00091             mp.add_arguments()
00092             mp.parser.print_help()
00093         exit()
00094
00095     def parse(self):
00096         markopy.MarkopyCLI.parse(self)
00097
00098     def parse_fail(self):
00099         "! @brief Not a valid model type"
00100         if(self.argsargsargsargs.model_type == "CUDA"):
00101             self.cliclicli = CudaModelMatrixCLI()
00102         else:
00103             markopy.MarkopyCLI.parse_fail(self)
00104
00105
00106 if __name__ == "__main__":
00107     mp = CudaMarkopyCLI()
00108     #mp = markopy_cli.MarkopyCLI()
00109     mp.parse()
00110     mp.process()
```

9.3 Markopy/CudaMarkopy/src/CLI/cudammx.py File Reference

CUDAModelMatrix CLI wrapper.

Classes

- class Python.CudaMarkopy.CudaModelMatrixCLI

  Python CLI wrapper for CudaModelMatrix.

Namespaces

- cudammx

Variables

- string cudammx.ext = "so"
- cudammx.spec = spec_from_loader("cudamarkopy", ExtensionFileLoader("cudamarkopy", os.path.↩ abspath(f"cudamarkopy.{ext}")))
- cudammx.cudamarkopy = module_from_spec(spec)
- cudammx.markopy = module_from_spec(spec)
- cudammx.mp = CudaModelMatrixCLI()

9.3.1 Detailed Description

CUDAModelMatrix CLI wrapper.
Definition in file cudammx.py.

9.4 cudammx.py

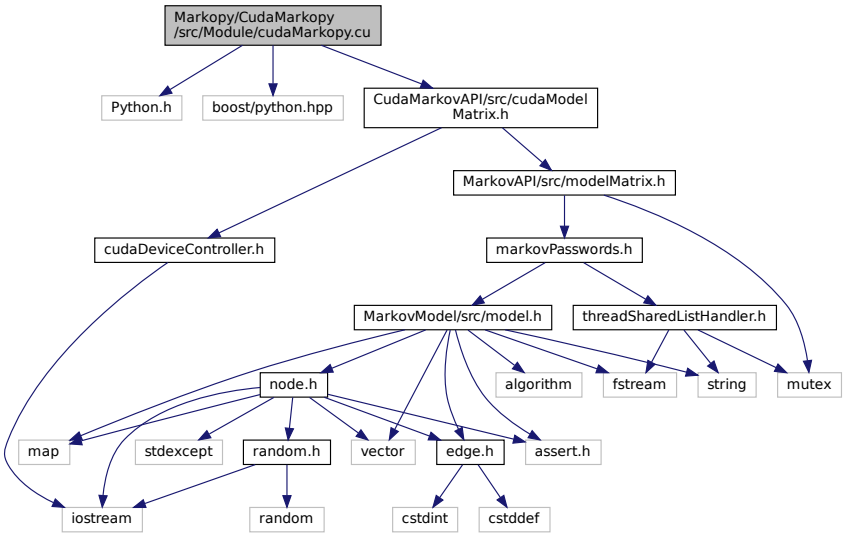
```
00001
00002
00006
00007
00008
00009 from importlib.util import spec_from_loader, module_from_spec
00010 from importlib.machinery import SourceFileLoader, ExtensionFileLoader
00011 import os
00012 import sys
00013 from mm import ModelMatrix
00014
00015 ext = "so"
00016 if os.name == 'nt':
00017     ext="pyd"
00018 try:
00019     spec = spec_from_loader("cudamarkopy", ExtensionFileLoader("cudamarkopy",
      os.path.abspath(f"cudamarkopy.{ext}")))
00020     cudamarkopy = module_from_spec(spec)
00021     spec.loader.exec_module(cudamarkopy)
00022 except ImportError as e:
00023     #print(f"({__file__}) Working in development mode. Trying to load cudamarkopy.{ext} from
      ../../../out/")
00024     if(os.path.exists(f"../../../out/lib/cudamarkopy.{ext}")):
00025         spec = spec_from_loader("cudamarkopy", ExtensionFileLoader("cudamarkopy",
      os.path.abspath(f"../../../out/lib/cudamarkopy.{ext}")))
00026         cudamarkopy = module_from_spec(spec)
00027         spec.loader.exec_module(cudamarkopy)
00028     else:
00029         raise e
00030
00031
00032
00033 try:
00034     spec = spec_from_loader("markopy", SourceFileLoader("markopy", os.path.abspath("markopy.py")))
00035     markopy = module_from_spec(spec)
00036
00037     from mmx import ModelMatrixCLI
00038     from base import BaseCLI,AbstractGenerationModelCLI, AbstractTrainingModelCLI
00039
00040 except ImportError as e:
00041     #print("Working in development mode. Trying to load from ../../../out/")
00042     if(os.path.exists("../../../Markopy/src/CLI/markopy.py")):
00043         spec = spec_from_loader("markopy", SourceFileLoader("markopy",
      os.path.abspath("../../../Markopy/src/CLI/markopy.py")))
00044         markopy = module_from_spec(spec)
00045         sys.path.insert(1, '../../../Markopy/src/CLI/')
00046
00047         from mmx import ModelMatrixCLI
00048         from base import BaseCLI,AbstractGenerationModelCLI, AbstractTrainingModelCLI
```

```
00049     else:
00050         raise e
00051
00052 import os
00053 import allogate as logging
00054
00055 class CudaModelMatrixCLI(ModelMatrixCLI,AbstractGenerationModelCLI):
00056     """!
00057         @belongsto Python::CudaMarkopy
00058         @brief Python CLI wrapper for CudaModelMatrix
00059         @extends Python::Markopy::ModelMatrixCLI
00060         @extends Python::Markopy::AbstractGenerationModelCLI
00061         @extends Markov::API::CUDA::CUDAModelMatrix
00062     """
00063     def __init__(self):
00064         super().__init__()
00065         self.modelmodelmodelmodel = cudamarkopy.CUDAModelMatrix()
00066
00067     def add_arguments(self):
00068         super().add_arguments()
00069         self.parserparser.add_argument("-if", "--infinite", action="store_true", help="Infinite
      generation mode")
00070
00071     def init_post_arguments(self):
00072         super().init_post_arguments()
00073         self.bInfinitebInfinite = self.argsargs.infinite
00074
00075     def _generate(self, wordlist : str ):
00076         self.modelmodelmodelmodel.FastRandomWalk(int(self.argsargs.count), wordlist,
      int(self.argsargs.min), int(self.argsargs.max), self.fileIOfileIO, self.bInfinitebInfinite)
00077
00078 if __name__ == "__main__":
00079     mp = CudaModelMatrixCLI()
00080     mp.parse()
00081     mp.process()
```

9.5 Markopy/CudaMarkopy/src/Module/cudaMarkopy.cu File Reference

```
#include <Python.h>
#include <boost/python.hpp>
#include "CudaMarkovAPI/src/cudaModelMatrix.h"
```

Include dependency graph for cudaMarkopy.cu:



Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

- Markov::Markopy

  CPython module for Markov::API objects.
- Markov::Markopy::CUDA

  CPython module for Markov::API::CUDA objects.

Macros

- #define BOOST_PYTHON_STATIC_LIB

Functions

- Markov::Markopy::CUDA::BOOST_PYTHON_MODULE (cudamarkopy)

9.5.1 Macro Definition Documentation

9.5.1.1 BOOST_PYTHON_STATIC_LIB

```
#define BOOST_PYTHON_STATIC_LIB
```

Definition at line 9 of file cudaMarkopy.cu.

9.6 cudaMarkopy.cu

```
00001 /** @file cudaMarkopy.cpp
00002  * @brief CPython wrapper for libcudamarkov utils. GPU
00003  * @authors Ata Hakçıl
00004  *
00005  * @copydoc markopy.cpp
00006  * @copydoc cudaModelMatrix.cu
00007  */
00008
00009 #define BOOST_PYTHON_STATIC_LIB
00010 #include <Python.h>
00011 #include <boost/python.hpp>
00012 #include "CudaMarkovAPI/src/cudaModelMatrix.h"
00013
00014 using namespace boost::python;
00015
00016 /**
00017  * @brief CPython module for Markov::API::CUDA objects
00018  */
00019 namespace Markov::Markopy::CUDA{
00020     BOOST_PYTHON_MODULE(cudamarkopy)
00021     {
00022         bool (Markov::API::MarkovPasswords::*Export)(const char*) = &Markov::Model<char>::Export;
00023         void (Markov::API::CUDA::CUDAModelMatrix::*FastRandomWalk)(unsigned long int, const char*,
       int, int, bool, bool) = &Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk;
00024
00025         class_<Markov::API::CUDA::CUDAModelMatrix>("CUDAModelMatrix", init<>())
00026
00027             .def(init<>())
00028             .def("Train", &Markov::API::ModelMatrix::Train)
00029             .def("Import", &Markov::API::ModelMatrix::Import, "Import a model file.")
00030             .def("Export", Export, "Export a model to file.")
00031             .def("ConstructMatrix",&Markov::API::ModelMatrix::ConstructMatrix)
00032             .def("DumpJSON",&Markov::API::ModelMatrix::DumpJSON)
00033             .def("FastRandomWalk", FastRandomWalk)
00034             ;
00035     };
00036 };
```

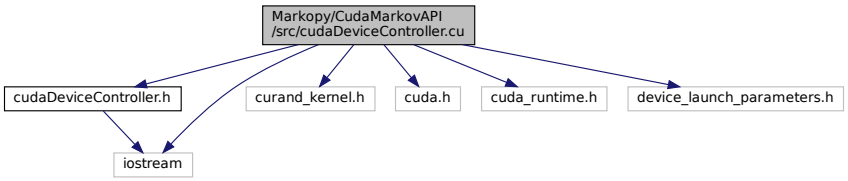
9.7 Markopy/CudaMarkovAPI/src/cudaDeviceController.cu File Reference

Simple static class for basic CUDA device controls.

```
#include "cudaDeviceController.h"
#include <iostream>
```

```
#include <curand_kernel.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <device_launch_parameters.h>
```

Include dependency graph for cudaDeviceController.cu:



Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.
- Markov::API::CUDA

  Namespace for objects requiring CUDA libraries.

9.7.1 Detailed Description

Simple static class for basic CUDA device controls.

Authors

Ata Hakçıl

Controller class for CUDA device. This implementation only supports Nvidia devices.
Definition in file cudaDeviceController.cu.

9.8 cudaDeviceController.cu

```
00001 /** @file cudaDeviceController.cu
00002  * @brief Simple static class for basic CUDA device controls.
00003  * @authors Ata Hakçıl
00004  *
00005  * @copydoc Markov::API::CUDA::CUDADeviceController
00006  */
00007
00008 #include "cudaDeviceController.h"
00009 #include <iostream>
00010 #include <curand_kernel.h>
00011 #include <cuda.h>
00012 #include <cuda_runtime.h>
00013 #include <device_launch_parameters.h>
00014
00015 namespace Markov::API::CUDA{
00016     __host__ void Markov::API::CUDA::CUDADeviceController::ListCudaDevices() { //list cuda Capable
       devices on host.
00017         int nDevices;
00018         cudaGetDeviceCount(&nDevices);
00019         for (int i = 0; i < nDevices; i++) {
00020             cudaDeviceProp prop;
00021             cudaGetDeviceProperties(&prop, i);
00022             std::cerr « "Device Number: " «  i  « "\n";
00023             std::cerr « "Device name: " « prop.name « "\n";
00024             std::cerr « "Memory Clock Rate (KHz): " « prop.memoryClockRate « "\n";
00025             std::cerr « "Memory Bus Width (bits): " « prop.memoryBusWidth « "\n";
00026             std::cerr « "Peak Memory Bandwidth (GB/s): " « 2.0 * prop.memoryClockRate *
       (prop.memoryBusWidth / 8) / 1.0e6 « "\n";
00027             std::cerr « "Max Linear Threads: " « prop.maxThreadsDim[0] « "\n";
```

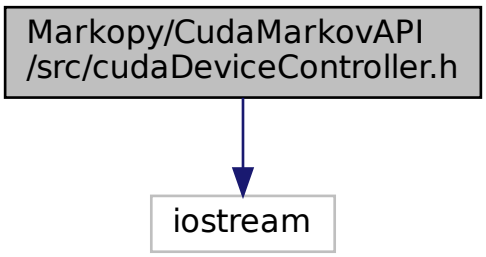
```
00028
00029         }
00030     }
00031
00032     __host__ int Markov::API::CUDA::CUDADeviceController::CudaCheckNotifyErr(cudaError_t _status,
      const char* msg, bool bExit) {
00033         if (_status != cudaSuccess) {
00034             std::cerr « "\033[1;31m" « msg  « " -> " « cudaGetErrorString(_status)  « " ("« _status «
      ")" « "\033[0m" « "\n";
00035
00036             if(bExit) {
00037                 cudaDeviceReset();
00038                 exit(1);
00039             }
00040         }
00041         return 0;
00042     }
00043
00044 /*
00045     template <typename T>
00046     __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat(T* dst, int row,
      int col){
00047         return  cudaMalloc((T **)&dst, row*col*sizeof(T));
00048     }
00049
00050     template <typename T>
00051     __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat(T* dst, T** src,
      int row, int col){
00052          cudaError_t cudastatus;
00053          for(int i=0;i<row;i++){
00054             cudastatus = cudaMemcpy(dst + (i*col*sizeof(T)),
00055                 src[i], col*sizeof(T), cudaMemcpyHostToDevice);
00056             if(cudastatus != cudaSuccess) return cudastatus;
00057         }
00058         return cudaSuccess;
00059     }
00060 */
00061
00062 };
```

9.9 Markopy/CudaMarkovAPI/src/cudaDeviceController.h File Reference

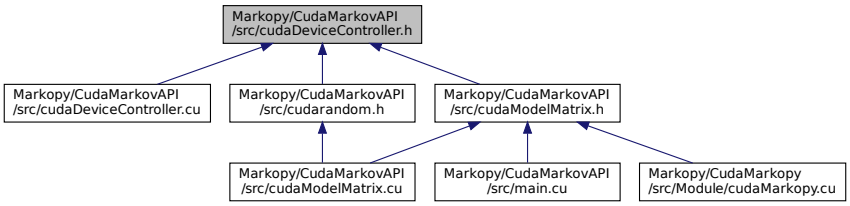
Simple static class for basic CUDA device controls.

`#include <iostream>`

Include dependency graph for cudaDeviceController.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::API::CUDA::CUDADeviceController

  Controller class for CUDA device.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.
- Markov::API::CUDA

  Namespace for objects requiring CUDA libraries.

9.9.1 Detailed Description

Simple static class for basic CUDA device controls.

Authors

Ata Hakçıl

Controller class for CUDA device. This implementation only supports Nvidia devices.
Definition in file cudaDeviceController.h.

9.10 cudaDeviceController.h

```
00001 /** @file cudaDeviceController.h
00002  * @brief Simple static class for basic CUDA device controls.
00003  * @authors Ata Hakçıl
00004  *
00005  * @copydoc Markov::API::CUDA::CUDADeviceController
00006  */
00007
00008 #pragma once
00009 #include <iostream>
00010
00011 /** @brief Namespace for objects requiring CUDA libraries.
00012 */
00013 namespace Markov::API::CUDA{
00014     /** @brief Controller class for CUDA device
00015      *
00016      * This implementation only supports Nvidia devices.
00017     */
00018     class CUDADeviceController{
00019     public:
00020         /** @brief List CUDA devices in the system.
00021          *
00022          * This function will print details of every CUDA capable device in the system.
00023          *
00024          * @b Example @b output:
00025          * @code{.txt}
00026          * Device Number: 0
00027          * Device name: GeForce RTX 2070
```

```
00028          * Memory Clock Rate (KHz): 7001000
00029          * Memory Bus Width (bits): 256
00030          * Peak Memory Bandwidth (GB/s): 448.064
00031          * Max Linear Threads: 1024
00032          * @endcode
00033         */
00034         __host__ static void ListCudaDevices();
00035
00036     protected:
00037         /** @brief Check results of the last operation on GPU.
00038          *
00039          * Check the status returned from cudaMalloc/cudaMemcpy to find failures.
00040          *
00041          * If a failure occurs, its assumed beyond redemption, and exited.
00042          * @param _status Cuda error status to check
00043          * @param msg Message to print in case of a failure
00044          * @return 0 if successful, 1 if failure.
00045          * @b Example @b output:
00046          * @code{.cpp}
00047          * char *da, a = "test";
00048          * cudastatus = cudaMalloc((char **)&da, 5*sizeof(char*));
00049          * CudaCheckNotifyErr(cudastatus, "Failed to allocate VRAM for *da.\n");
00050          * @endcode
00051         */
00052         __host__ static int CudaCheckNotifyErr(cudaError_t _status, const char* msg, bool bExit=true);
00053
00054
00055         /** @brief Malloc a 2D array in device space
00056          *
00057          * This function will allocate enough space on VRAM for flattened 2D array.
00058          *
00059          * @param dst destination pointer
00060          * @param row row size of the 2d array
00061          * @param col column size of the 2d array
00062          * @return cudaError_t status of the cudaMalloc operation
00063          *
00064          * @b Example @b output:
00065          * @code{.cpp}
00066          *   cudaError_t cudastatus;
00067          *   char* dst;
00068          *   cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
00069          *   if(cudastatus!=cudaSuccess){
00070          *       CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
00071          *   }
00072          * @endcode
00073         */
00074         template <typename T>
00075         __host__ static cudaError_t CudaMalloc2DToFlat(T** dst, int row, int col){
00076             cudaError_t cudastatus = cudaMalloc((T **)dst, row*col*sizeof(T));
00077             CudaCheckNotifyErr(cudastatus, "cudaMalloc Failed.", false);
00078             return cudastatus;
00079         }
00080
00081
00082         /** @brief Memcpy a 2D array in device space after flattening
00083          *
00084          * Resulting buffer will not be true 2D array.
00085          *
00086          * @param dst destination pointer
00087          * @param rc  source pointer
00088          * @param row row size of the 2d array
00089          * @param col column size of the 2d array
00090          * @return cudaError_t status of the cudaMalloc operation
00091          *
00092          * @b Example @b output:
00093          * @code{.cpp}
00094          *   cudaError_t cudastatus;
00095          *   char* dst;
00096          *   cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
00097          *   CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
00098          *   cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
00099          *   CudaCheckNotifyErr(cudastatus, "  CudaMemcpy2DToFlat Failed.", false);
00100          * @endcode
00101         */
00102         template <typename T>
00103         __host__ static cudaError_t CudaMemcpy2DToFlat(T* dst, T** src, int row, int col){
00104             T* tempbuf = new T[row*col];
00105             for(int i=0;i<row;i++){
00106                 memcpy(&(tempbuf[row*i]), src[i], col);
00107             }
00108             return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00109
00110         }
00111
00112         /** @brief Both malloc and memcpy a 2D array into device VRAM.
00113          *
00114          * Resulting buffer will not be true 2D array.
```

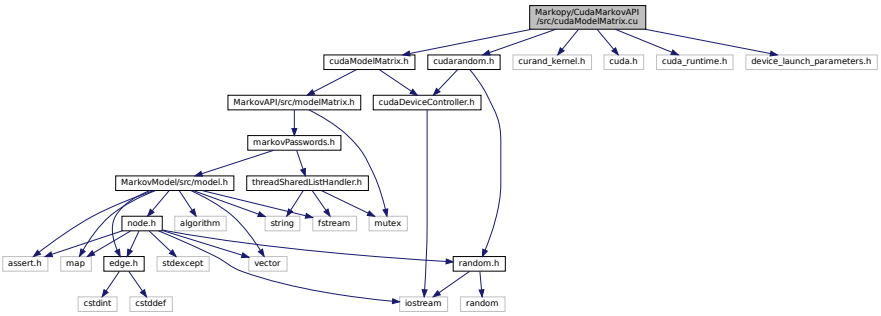
```
00115          *
00116          * @param dst destination pointer
00117          * @param rc  source pointer
00118          * @param row row size of the 2d array
00119          * @param col column size of the 2d array
00120          * @return cudaError_t status of the cudaMalloc operation
00121          *
00122          * @b Example @b output:
00123          * @code{.cpp}
00124          *   cudaError_t cudastatus;
00125          *   char* dst;
00126          *   cudastatus = CudaMigrate2DFlat<long int>(
00127          *      &dst, this->valueMatrix, this->matrixSize, this->matrixSize);
00128          *   CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
00129          * @endcode
00130         */
00131         template <typename T>
00132         __host__ static cudaError_t CudaMigrate2DFlat(T** dst, T** src, int row, int col){
00133             cudaError_t cudastatus;
00134             cudastatus = CudaMalloc2DToFlat<T>(dst, row, col);
00135             if(cudastatus!=cudaSuccess){
00136                 CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
00137                 return cudastatus;
00138             }
00139             cudastatus = CudaMemcpy2DToFlat<T>(*dst,src,row,col);
00140             CudaCheckNotifyErr(cudastatus, "  CudaMemcpy2DToFlat Failed.", false);
00141             return cudastatus;
00142         }
00143
00144
00145     private:
00146     };
00147 };
```

9.11 Markopy/CudaMarkovAPI/src/cudaModelMatrix.cu File Reference

CUDA accelerated extension of Markov::API::ModelMatrix.

```
#include "cudaModelMatrix.h"
#include "cudarandom.h"
#include <curand_kernel.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <device_launch_parameters.h>
```

Include dependency graph for cudaModelMatrix.cu:



Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

- Markov::API

  Namespace for the MarkovPasswords API.

- Markov::API::CUDA

  Namespace for objects requiring CUDA libraries.

Functions

- __global__ void Markov::API::CUDA::FastRandomWalkCUDAKernel (unsigned long int n, int minLen, int maxLen, char *outputBuffer, char *matrixIndex, long int *totalEdgeWeights, long int *valueMatrix, char *edgeMatrix, int matrixSize, int memoryPerKernelGrid, unsigned long *seed)

  CUDA kernel for the FastRandomWalk operation.
- __device__ char * Markov::API::CUDA::strchr (char *p, char c, int s_len)

  *srtchr implementation on **device** space*

9.11.1 Detailed Description

CUDA accelerated extension of Markov::API::ModelMatrix.

Authors

Ata Hakçıl

Extension of Markov::API::ModelMatrix which is modified to run on GPU devices. This implementation only supports Nvidia devices.
Class to flatten and reduce Markov::Model to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)
To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are sychronized and files are flushed every 50M operations.
Definition in file cudaModelMatrix.cu.

9.12 cudaModelMatrix.cu

```
00001 /** @file cudaModelMatrix.cu
00002  * @brief CUDA accelerated extension of Markov::API::ModelMatrix
00003  * @authors Ata Hakçıl
00004  *
00005  * @copydoc Markov::API::CUDA::CUDAModelMatrix
00006  */
00007
00008 #include "cudaModelMatrix.h"
00009 #include "cudarandom.h"
00010
00011
00012 #include <curand_kernel.h>
00013 #include <cuda.h>
00014 #include <cuda_runtime.h>
00015 #include <device_launch_parameters.h>
00016
00017 using Markov::API::CUDA::CUDADeviceController;
00018
00019 namespace Markov::API::CUDA{
00020     __host__ void Markov::API::CUDA::CUDAModelMatrix::MigrateMatrix(){
00021         cudaError_t cudastatus;
00022
00023         cudastatus = cudaMalloc((char**)&(this->device_matrixIndex),
00024             this->matrixSize*sizeof(char));
00025         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_matrixIndex.");
00026
00027         cudastatus = cudaMalloc((long int **)&(this->device_totalEdgeWeights),
      this->matrixSize*sizeof(long int));
00028         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_totalEdgeWeights.");
00029
00030         cudastatus = cudaMemcpy(this->device_matrixIndex, this->matrixIndex,
00031             this->matrixSize*sizeof(char), cudaMemcpyHostToDevice);
00032         CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Index)");
00033
00034         cudastatus = cudaMemcpy(this->device_totalEdgeWeights, this->totalEdgeWeights,
00035             this->matrixSize*sizeof(long int), cudaMemcpyHostToDevice);
00036         CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Total Edge Values)");
00037
00038         cudastatus = CudaMigrate2DFlat<char>(
00039             &(this->device_edgeMatrix), this->edgeMatrix, this->matrixSize, this->matrixSize);
00040         CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize edge matrix.");
00041
00042         cudastatus = CudaMigrate2DFlat<long int>(
00043             &(this->device_valueMatrix), this->valueMatrix, this->matrixSize, this->matrixSize);
00044         CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
00045
```

```
00046     }
00047
00048     /*__host__ char*  Markov::API::CUDA::CUDAModelMatrix::AllocVRAMOutputBuffer(long int n, long int
      singleGenMaxLen, long int CUDAKernelGridSize,long int sizePerGrid){
00049         cudaError_t cudastatus;
00050         cudastatus = cudaMalloc((char **)&this->device_outputBuffer1, CUDAKernelGridSize*sizePerGrid);
00051         CudaCheckNotifyErr(cudastatus, "Failed to allocate VRAM buffer. (Possibly out of VRAM.)");
00052
00053         return this->device_outputBuffer1;
00054     }*/
00055
00056
00057
00058     __host__ void Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk(unsigned long int n, const char*
      wordlistFileName, int minLen, int maxLen, bool bFileIO, bool bInfinite){
00059         cudaDeviceProp prop;
00060         int device=0;
00061         cudaGetDeviceProperties(&prop, device);
00062         cudaChooseDevice(&device, &prop);
00063         //std::cout << "Flattening matrix." << std::endl;
00064         this->FlattenMatrix();
00065         //std::cout << "Migrating matrix." << std::endl;
00066         this->MigrateMatrix();
00067         //std::cout << "Migrated matrix." << std::endl;
00068         std::ofstream wordlist;
00069         if(bFileIO)
00070             wordlist.open(wordlistFileName);
00071
00072
00073         cudaBlocks = 1024;
00074         cudaThreads = 256;
00075         iterationsPerKernelThread = 100;
00076         alternatingKernels = 2;
00077         totalOutputPerKernel = (long int)cudaBlocks*(long int)cudaThreads*iterationsPerKernelThread;
00078         totalOutputPerSync= totalOutputPerKernel*alternatingKernels;
00079         numberOfPartitions = n/totalOutputPerSync;
00080         cudaGridSize = cudaBlocks*cudaThreads;
00081         cudaMemPerGrid = (maxLen+2)*iterationsPerKernelThread;
00082         cudaPerKernelAllocationSize = cudaGridSize*cudaMemPerGrid;
00083         this->prepKernelMemoryChannel(alternatingKernels);
00084
00085         unsigned long int leftover = n - (totalOutputPerSync*numberOfPartitions);
00086
00087         if(bInfinite && !numberOfPartitions) numberOfPartitions=5;
00088         std::cerr << cudaPerKernelAllocationSize << "\n";
00089
00090         if(n%totalOutputPerSync) std::cerr << "For optimization, request outputs muliples of "<<
      totalOutputPerSync << ".\n";
00091
00092         //start kernelID 1
00093         this->LaunchAsyncKernel(1, minLen, maxLen);
00094
00095         for(int i=1;i<numberOfPartitions;i++){
00096             if(bInfinite) i=0;
00097
00098             //wait kernelID1 to finish, and start kernelID 0
00099             cudaStreamSynchronize(this->cudastreams[1]);
00100             this->LaunchAsyncKernel(0, minLen, maxLen);
00101
00102             //start memcpy from kernel 1 (block until done)
00103             this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00104
00105             //wait kernelID 0 to finish, then start kernelID1
00106             cudaStreamSynchronize(this->cudastreams[0]);
00107             this->LaunchAsyncKernel(1, minLen, maxLen);
00108
00109             //start memcpy from kernel 0 (block until done)
00110             this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00111
00112         }
00113
00114         //wait kernelID1 to finish, and start kernelID 0
00115         cudaStreamSynchronize(this->cudastreams[1]);
00116         this->LaunchAsyncKernel(0, minLen, maxLen);
00117         this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00118         cudaStreamSynchronize(this->cudastreams[0]);
00119         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00120
00121
00122         if(!leftover) return;
00123         alternatingKernels=1;
00124         std::cerr << "Remaining line count (" << leftover << ") is lower than partition. Adjusting CUDA
      workload..\n";
00125         this->iterationsPerKernelThread = leftover/cudaGridSize;
00126         this->LaunchAsyncKernel(0, minLen, maxLen);
00127         cudaStreamSynchronize(this->cudastreams[0]);
00128         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
```

```
00129
00130         leftover -= this->iterationsPerKernelThread*cudaGridSize;
00131         if(!leftover) return;
00132
00133         std::cerr « "Remaining line count (" « leftover « ") is lower than minimum possible. Handing
      over to CPU generation.\n";
00134         this->iterationsPerKernelThread = leftover/cudaGridSize;
00135
00136         leftover -= this->iterationsPerKernelThread;
00137
00138         if(!leftover) return;
00139         std::cerr « "Remaining " « leftover « " lines are absolutely not worth printing.\n";
00140         Markov::API::ModelMatrix::ConstructMatrix();
00141         Markov::API::ModelMatrix::FastRandomWalk(leftover, &wordlist, minLen, maxLen, 1, bFileIO);
00142
00143     }
00144
00145     __host__ void Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel(int numberOfStreams){
00146
00147         this->cudastreams = new cudaStream_t[numberOfStreams];
00148         for(int i=0;i<numberOfStreams;i++)
00149             cudaStreamCreate(&this->cudastreams[i]);
00150
00151         this-> outputBuffer = new char*[numberOfStreams];
00152         for(int i=0;i<numberOfStreams;i++)
00153             this->outputBuffer[i]= new char[cudaPerKernelAllocationSize];
00154
00155         cudaError_t cudastatus;
00156         this-> device_outputBuffer = new char*[numberOfStreams];
00157             for(int i=0;i<numberOfStreams;i++){
00158                 cudastatus = cudaMalloc((char**)&(device_outputBuffer[i]),
      cudaPerKernelAllocationSize);
00159                 CudaCheckNotifyErr(cudastatus, "Failed to establish memory channel. Possibly out of
      VRAM?");
00160             }
00161
00162         this-> device_seeds = new unsigned long*[numberOfStreams];
00163         for(int i=0;i<numberOfStreams;i++){
00164             Markov::API::CUDA::Random::Marsaglia *MEarr = new
      Markov::API::CUDA::Random::Marsaglia[cudaGridSize];
00165             this->device_seeds[i] = Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(MEarr,
      cudaGridSize);
00166             delete[] MEarr;
00167         }
00168
00169     }
00170
00171     __host__ void Markov::API::CUDA::CUDAModelMatrix::LaunchAsyncKernel(int kernelID, int minLen, int
      maxLen){
00172
00173         //if(kernelID == 0);// cudaStreamSynchronize(this->cudastreams[2]);
00174         //else cudaStreamSynchronize(this->cudastreams[kernelID-1]);
00175         FastRandomWalkCUDAKernel«<cudaBlocks,cudaThreads,0,
      this->cudastreams[kernelID]»>(iterationsPerKernelThread, minLen, maxLen,
      this->device_outputBuffer[kernelID], this->device_matrixIndex,
00176             this->device_totalEdgeWeights, this->device_valueMatrix, this->device_edgeMatrix,
      this->matrixSize, cudaMemPerGrid, this->device_seeds[kernelID]);
00177         //std::cerr « "Started kernel" « kernelID « "\n";
00178     }
00179
00180     __host__ void Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput(int kernelID, bool
      bFileIO, std::ofstream &wordlist){
00181
      cudaMemcpy(this->outputBuffer[kernelID],this->device_outputBuffer[kernelID],cudaPerKernelAllocationSize,
      cudaMemcpyDeviceToHost);
00182         //std::cerr « "Kernel" « kernelID « " output copied\n";
00183         if(bFileIO){
00184             for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00185                 wordlist « &this->outputBuffer[kernelID][j];
00186             }
00187         }else{
00188             for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00189                 std::cout « &this->outputBuffer[kernelID][j];
00190             }
00191         }
00192     }
00193
00194     __global__ void FastRandomWalkCUDAKernel(unsigned long int n, int minLen, int maxLen, char*
      outputBuffer,
00195     char* matrixIndex, long int* totalEdgeWeights, long int* valueMatrix, char *edgeMatrix, int
      matrixSize, int memoryPerKernelGrid, unsigned long *seed){
00196
00197         int kernelWorkerIndex = threadIdx.x + blockIdx.x * blockDim.x;
00198
00199         if(n==0) return;
00200
00201         char* e;
```

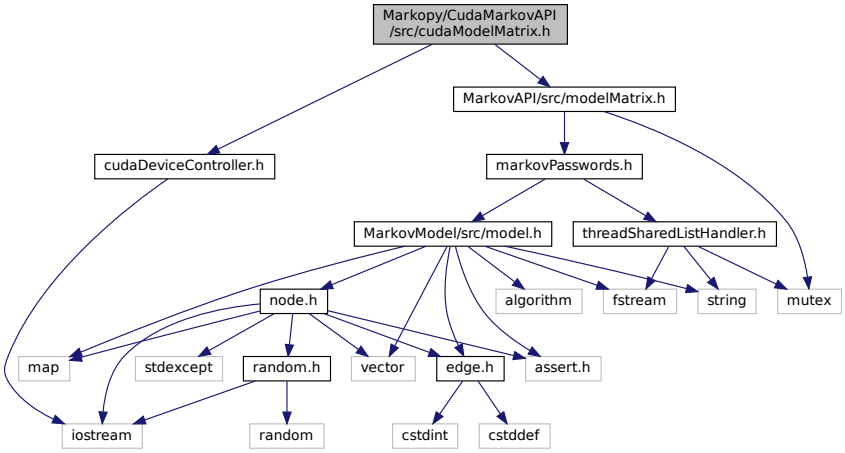
```
00202         int index = 0;
00203         char next;
00204         int len=0;
00205         long int selection;
00206         char cur;
00207         long int bufferctr = 0;
00208         unsigned long int *x,*y,*z,t;
00209         char* res = &outputBuffer[kernelWorkerIndex*memoryPerKernelGrid];
00210         x=&seed[kernelWorkerIndex*3];
00211         y=&seed[kernelWorkerIndex*3+1];
00212         z=&seed[kernelWorkerIndex*3+2];
00213         for (int i = 0; i < n; i++) {
00214             cur=199;
00215             len=0;
00216             while (true) {
00217                 e = strchr(matrixIndex, cur, matrixSize);
00218                 index = e - matrixIndex;
00219                 /*selection = Markov::API::CUDA::Random::devrandom(
00220                     seed[kernelWorkerIndex*3],
00221                     seed[kernelWorkerIndex*3+1],
00222                     seed[kernelWorkerIndex*3+2]) % totalEdgeWeights[index];*/
00223                 *x ^= *x « 16;
00224                 *x ^= *x » 5;
00225                 *x ^= *x « 1;
00226 
00227                 t = *x;
00228                 *x = *y;
00229                 *y = *z;
00230                 *z = t ^ *x ^ *y;
00231                 selection = *z % totalEdgeWeights[index];
00232             for(int j=0;j<matrixSize-1;j++){
00233                     selection -= valueMatrix[index*matrixSize + j];
00234                     if (selection < 0){
00235                         next = edgeMatrix[index*sizeof(char)*matrixSize + j];
00236                         break;
00237                     }
00238                 }
00239 
00240                 if (len >= maxLen)  break;
00241                 else if ((next < 0) && (len < minLen)) continue;
00242                 else if (next < 0) break;
00243                 cur = next;
00244                 res[bufferctr + len++] = cur;
00245             }
00246             res[bufferctr + len++] = '\n';
00247             bufferctr+=len;
00248         }
00249         res[bufferctr] = '\0';
00250     }
00251 
00252     __device__ char* strchr(char* p, char c, int s_len){
00253        for (;; ++p, s_len--) {
00254             if (*p ==  c)
00255                 return((char *)p);
00256             if (!*p)
00257                 return((char *)NULL);
00258         }
00259     }
00260 
00261     __host__ void Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix(){
00262         this->flatEdgeMatrix = new char[this->matrixSize*this->matrixSize];
00263 
00264         this->flatValueMatrix = new long int[this->matrixSize*this->matrixSize];
00265         for(int i=0;i<this->matrixSize;i++){
00266             memcpy(&this->flatEdgeMatrix[i*this->matrixSize], this->edgeMatrix[i], this->matrixSize );
00267             memcpy(&this->flatValueMatrix[i*this->matrixSize], this->valueMatrix[i],
      this->matrixSize*sizeof(long int) );
00268         }
00269     }
00270 
00271 
00272 };
```

9.13 Markopy/CudaMarkovAPI/src/cudaModelMatrix.h File Reference

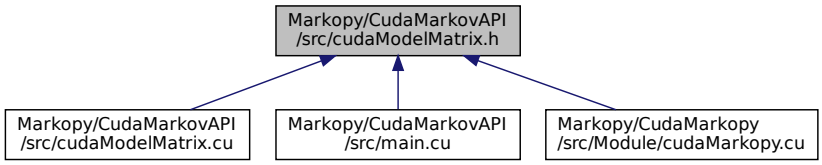
CUDA accelerated extension of Markov::API::ModelMatrix.

```
#include "MarkovAPI/src/modelMatrix.h"
#include "cudaDeviceController.h"
```

Include dependency graph for cudaModelMatrix.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::API::CUDA::CUDAModelMatrix

  Extension of Markov::API::ModelMatrix which is modified to run on GPU devices.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.
- Markov::API::CUDA

  Namespace for objects requiring CUDA libraries.

Functions

- __global__ void Markov::API::CUDA::FastRandomWalkCUDAKernel (unsigned long int n, int minLen, int maxLen, char ∗outputBuffer, char ∗matrixIndex, long int ∗totalEdgeWeights, long int ∗valueMatrix, char ∗edgeMatrix, int matrixSize, int memoryPerKernelGrid, unsigned long ∗seed)

  CUDA kernel for the FastRandomWalk operation.
- __device__ char ∗ Markov::API::CUDA::strchr (char ∗p, char c, int s_len)

  *srtchr implementation on **device** space*

9.13.1 Detailed Description

CUDA accelerated extension of Markov::API::ModelMatrix.

Authors

Ata Hakçıl

Extension of Markov::API::ModelMatrix which is modified to run on GPU devices. This implementation only supports Nvidia devices.
Class to flatten and reduce Markov::Model to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)
To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are sychronized and files are flushed every 50M operations.
Definition in file cudaModelMatrix.h.

9.14 cudaModelMatrix.h

```
00001 /** @file cudaModelMatrix.h
00002  * @brief CUDA accelerated extension of Markov::API::ModelMatrix
00003  * @authors Ata Hakçıl
00004  *
00005  * @copydoc Markov::API::CUDA::CUDAModelMatrix
00006  */
00007
00008 #include "MarkovAPI/src/modelMatrix.h"
00009 #include "cudaDeviceController.h"
00010
00011 /** @brief Namespace for objects requiring CUDA libraries.
00012 */
00013 namespace Markov::API::CUDA{
00014     /** @brief Extension of Markov::API::ModelMatrix which is modified to run on GPU devices.
00015      *
00016      * This implementation only supports Nvidia devices.
00017      * @copydoc Markov::API::ModelMatrix
00018     */
00019     class CUDAModelMatrix : public Markov::API::ModelMatrix, public CUDADeviceController{
00020     public:
00021
00022         /** @brief Migrate the class members to the VRAM
00023          *
00024          * Cannot be used without calling Markov::API::ModelMatrix::ConstructMatrix at least once.
00025          * This function will manage the memory allocation and data transfer from CPU RAM to GPU VRAM.
00026          *
00027          * Newly allocated VRAM pointers are set in the class member variables.
00028          *
00029         */
00030         __host__ void MigrateMatrix();
00031
00032         /** @brief Flatten migrated matrix from 2d to 1d
00033          *
00034          *
00035         */
00036         __host__ void FlattenMatrix();
00037
00038         /** @brief Random walk on the Matrix-reduced Markov::Model
00039          *
00040          * TODO
00041          *
00042          *
00043          * @param n - Number of passwords to generate.
00044          * @param wordlistFileName - Filename to write to
00045          * @param minLen - Minimum password length to generate
00046          * @param maxLen - Maximum password length to generate
00047          * @param threads - number of OS threads to spawn
00048          * @param bFileIO - If false, filename will be ignored and will output to stdout.
00049          *
00050          *
00051          * @code{.cpp}
00052          * Markov::API::ModelMatrix mp;
00053          * mp.Import("models/finished.mdl");
00054          * mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
00055          * @endcode
00056          *
00057         */
00058         __host__ void FastRandomWalk(unsigned long int n, const char* wordlistFileName, int minLen,
      int maxLen, bool bFileIO, bool bInfinite);
00059
00060     protected:
```

```
00061
00062         /** @brief Allocate the output buffer for kernel operation
00063          *
00064          * TODO
00065          *
00066          *
00067          * @param n - Number of passwords to generate.
00068          * @param singleGenMaxLen - maximum string length for a single generation
00069          * @param CUDAKernelGridSize - Total number of grid members in CUDA kernel
00070          * @param sizePerGrid - Size to allocate per grid member
00071          * @return pointer to the allocation on VRAM
00072          *
00073          *
00074         */
00075         __host__ char* AllocVRAMOutputBuffer(long int n, long int singleGenMaxLen, long int
      CUDAKernelGridSize,long int sizePerGrid);
00076
00077         __host__ void LaunchAsyncKernel(int kernelID, int minLen, int maxLen);
00078
00079         __host__ void prepKernelMemoryChannel(int numberOfStreams);
00080
00081         __host__ void GatherAsyncKernelOutput(int kernelID, bool bFileIO, std::ofstream &wordlist);
00082
00083     private:
00084
00085         /**
00086             @brief VRAM Address pointer of edge matrix (from modelMatrix.h)
00087         */
00088         char* device_edgeMatrix;
00089
00090         /**
00091             @brief VRAM Address pointer of value matrix (from modelMatrix.h)
00092         */
00093         long int *device_valueMatrix;
00094
00095         /**
00096             @brief VRAM Address pointer of matrixIndex (from modelMatrix.h)
00097         */
00098         char *device_matrixIndex;
00099
00100         /**
00101             @brief VRAM Address pointer of total edge weights (from modelMatrix.h)
00102         */
00103         long int *device_totalEdgeWeights;
00104
00105         /**
00106             @brief RandomWalk results in device
00107         */
00108         char** device_outputBuffer;
00109
00110         /**
00111             @brief RandomWalk  results in host
00112         */
00113         char** outputBuffer;
00114
00115         /**
00116             @brief Adding Edge matrix end-to-end and resize to 1-D array for better perfomance on
      traversing
00117         */
00118         char* flatEdgeMatrix;
00119
00120         /**
00121             @brief Adding Value matrix end-to-end and resize to 1-D array for better perfomance on
      traversing
00122         */
00123         long int* flatValueMatrix;
00124
00125         int cudaBlocks;
00126         int cudaThreads;
00127         int iterationsPerKernelThread;
00128         long int totalOutputPerSync;
00129         long int totalOutputPerKernel;
00130         int numberOfPartitions;
00131         int cudaGridSize;
00132         int cudaMemPerGrid;
00133         long int cudaPerKernelAllocationSize;
00134
00135         int alternatingKernels;
00136
00137         unsigned long**  device_seeds;
00138
00139         cudaStream_t *cudastreams;
00140
00141     };
00142
00143     /** @brief CUDA kernel for the FastRandomWalk operation
00144      *
```

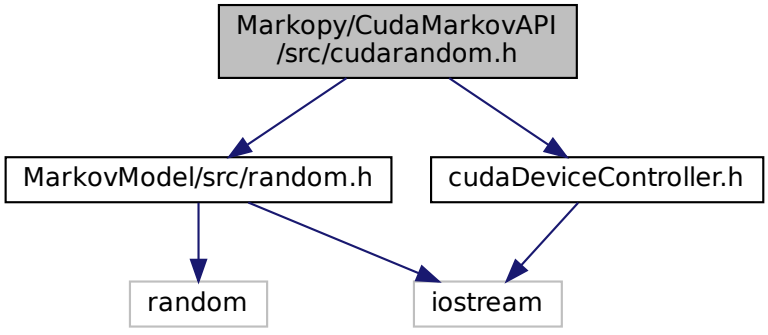
```
00145      * Will be initiated by CPU and continued by GPU (__global__ tag)
00146      *
00147      *
00148      * @param n - Number of passwords to generate.
00149      * @param minlen - minimum string length for a single generation
00150      * @param maxLen - maximum string length for a single generation
00151      * @param outputBuffer - VRAM ptr to the output buffer
00152      * @param matrixIndex - VRAM ptr to the matrix indices
00153      * @param totalEdgeWeights - VRAM ptr to the totalEdgeWeights array
00154      * @param valueMatrix - VRAM ptr to the edge weights array
00155      * @param edgeMatrix - VRAM ptr to the edge representations array
00156      * @param matrixSize - Size of the matrix dimensions
00157      * @param memoryPerKernelGrid - Maximum memory usage per kernel grid
00158      * @param seed - seed chunk to generate the random from (generated & used by Marsaglia)
00159      *
00160      *
00161      *
00162     */
00163     __global__ void FastRandomWalkCUDAKernel(unsigned long int n, int minLen, int maxLen, char*
      outputBuffer,
00164         char* matrixIndex, long int* totalEdgeWeights, long int* valueMatrix, char *edgeMatrix,
00165         int matrixSize, int memoryPerKernelGrid, unsigned long *seed);//, unsigned long mex, unsigned
      long mey, unsigned long mez);
00166
00167
00168     /** @brief srtchr implementation on __device__ space
00169      *
00170      * Fint the first matching index of a string
00171      *
00172      *
00173      * @param p - string to check
00174      * @param c - character to match
00175      * @param s_len - maximum string length
00176      * @returns pointer to the match
00177     */
00178     __device__ char* strchr(char* p, char c, int s_len);
00179
00180 };
```

9.15 Markopy/CudaMarkovAPI/src/cudarandom.h File Reference

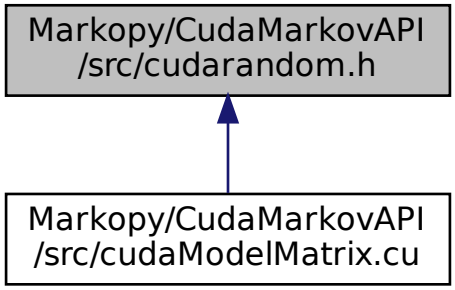
Extension of Markov::Random::Marsaglia for CUDA.

```
#include "MarkovModel/src/random.h"
#include "cudaDeviceController.h"
```

Include dependency graph for cudarandom.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::API::CUDA::Random::Marsaglia

  *Extension of Markov::Random::Marsaglia which is capable o working on **device** space.*

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.
- Markov::API::CUDA

  Namespace for objects requiring CUDA libraries.
- Markov::API::CUDA::Random

  *Namespace for Random engines operable under **device** space.*

Functions

- __device__ unsigned long Markov::API::CUDA::Random::devrandom (unsigned long &x, unsigned long &y, unsigned long &z)

  *Marsaglia Random Generation function operable in **device** space.*

9.15.1 Detailed Description

Extension of Markov::Random::Marsaglia for CUDA.

Authors

Ata Hakçıl

Implementation of Marsaglia Random Engine. This is an implementation of Marsaglia Random engine, which for most use cases is a better fit than other solutions. Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.
This implementation of the Marsaglia Engine is seeded by random.h default random engine. RandomEngine is only seeded once so its not a performance issue.
Example Use: Using Marsaglia Engine with RandomWalk

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
```

```
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout « res « "\n";
 }
```

Example Use: Generating a random number with Marsaglia Engine

```
Markov::Random::Marsaglia me;
std::cout « me.random();
```

Definition in file cudarandom.h.

9.16 cudarandom.h

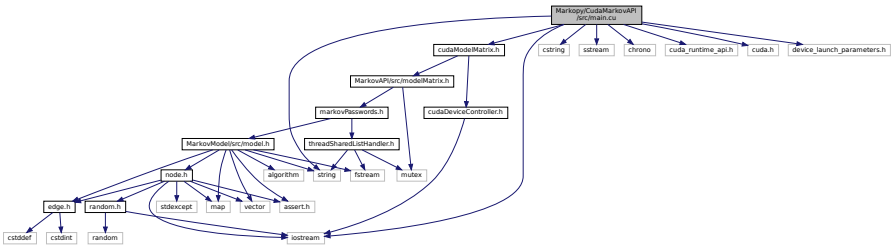
```
00001 /** @file cudarandom.h
00002  * @brief Extension of Markov::Random::Marsaglia for CUDA
00003  * @authors Ata Hakçıl
00004  *
00005  * @copydoc Markov::Random::Marsaglia
00006  */
00007
00008 #pragma once
00009 #include "MarkovModel/src/random.h"
00010 #include "cudaDeviceController.h"
00011
00012 /** @brief Namespace for Random engines operable under __device__ space.
00013 */
00014 namespace Markov::API::CUDA::Random{
00015
00016     /** @brief Extension of Markov::Random::Marsaglia which is capable o working on __device__ space.
00017      *
00018      * @copydoc Markov::Random::Marsaglia
00019     */
00020     class Marsaglia : public Markov::Random::Marsaglia, public CUDADeviceController{
00021     public:
00022
00023         /** @brief Migrate a Marsaglia[] to VRAM as seedChunk
00024          * @param MEarr Array of Marsaglia Engines
00025          * @param gridSize GridSize of the CUDA Kernel, aka size of array
00026          * @returns pointer to the resulting seed chunk in device VRAM.
00027         */
00028         static unsigned long* MigrateToVRAM(Markov::API::CUDA::Random::Marsaglia *MEarr, long int
      gridSize){
00029             cudaError_t cudastatus;
00030             unsigned long* seedChunk;
00031             cudastatus = cudaMalloc((unsigned long**)&seedChunk, gridSize*3*sizeof(unsigned long));
00032             CudaCheckNotifyErr(cudastatus, "Failed to allocate seed buffer");
00033             unsigned long *temp = new unsigned long[gridSize*3];
00034             for(int i=0;i<gridSize;i++){
00035                 temp[i*3]   = MEarr[i].x;
00036                 temp[i*3+1] = MEarr[i].y;
00037                 temp[i*3+2] = MEarr[i].z;
00038             }
00039             //for(int i=0;i<gridSize*3;i++) std::cout « temp[i] « "\n";
00040             cudaMemcpy(seedChunk, temp, gridSize*3*sizeof(unsigned long), cudaMemcpyHostToDevice);
00041             CudaCheckNotifyErr(cudastatus, "Failed to memcpy seed buffer.");
00042
00043             delete[] temp;
00044             return seedChunk;
00045         }
00046     };
00047
00048     /** @brief Marsaglia Random Generation function operable in __device__ space
00049      * @param x marsaglia internal x. Not constant, (ref)
00050      * @param y marsaglia internal y. Not constant, (ref)
00051      * @param z marsaglia internal z. Not constant, (ref)
00052      * @returns returns z
00053     */
00054     __device__ unsigned long devrandom(unsigned long &x, unsigned long &y, unsigned long &z){
00055         unsigned long t;
00056         x ^= x « 16;
00057         x ^= x » 5;
00058         x ^= x « 1;
00059
00060         t = x;
00061         x = y;
00062         y = z;
00063         z = t ^ x ^ y;
00064
00065         return z;
00066     }
00067 };
```

9.17 Markopy/CudaMarkovAPI/src/main.cu File Reference

Simple test file to check libcudamarkov.

```
#include <iostream>
#include <string>
#include <cstring>
#include <sstream>
#include <chrono>
#include "cudaModelMatrix.h"
#include <cuda_runtime_api.h>
#include <cuda.h>
#include <device_launch_parameters.h>
```

Include dependency graph for main.cu:



Functions

- int main (int argc, char **argv)

9.17.1 Detailed Description

Simple test file to check libcudamarkov.

Authors

Ata Hakçıl

Controller class for CUDA device. This implementation only supports Nvidia devices.
Definition in file main.cu.

9.17.2 Function Documentation

9.17.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

Definition at line 21 of file main.cu.

```
00021                                {
00022
00023
00024
00025     Markov::API::CUDA::CUDAModelMatrix markovPass;
00026     std::cerr « "Importing model.\n";
00027     markovPass.Import("models/finished.mdl");
00028     markovPass.Buff("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~", 10, true, true);
00029     std::cerr « "Import done. \n";
00030     markovPass.ConstructMatrix();
00031     //markovPass.DumpJSON();
00032     CUDADeviceController::ListCudaDevices();
00033
00034     std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
00035     std::cerr « "Starting walk. \n";
```

```
00036     markovPass.FastRandomWalk(1000000000,"/media/ignis/Stuff/wordlist.txt",12,12, false, false);
00037     //markovPass.FastRandomWalk(500000000,"/media/ignis/Stuff/wordlist2.txt",6,12,25, true);
00038     std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
00039
00040     std::cerr « "Finished in:" « std::chrono::duration_cast<std::chrono::milliseconds> (end -
       begin).count() « " milliseconds" « std::endl;
00041
00042
00043 }
```

9.18 main.cu

```
00001 /** @file main.cu
00002  * @brief Simple test file to check libcudamarkov
00003  * @authors Ata Hakçıl
00004  *
00005  * @copydoc Markov::API::CUDA::CudaModelMatrix
00006  * @copydoc Markov::API::CUDA::CUDADeviceController
00007  */
00008
00009 #include <iostream>
00010 #include <string>
00011 #include <cstring>
00012 #include <sstream>
00013 #include <chrono>
00014 #include "cudaModelMatrix.h"
00015 #include <cuda_runtime_api.h>
00016 #include <cuda.h>
00017 #include <device_launch_parameters.h>
00018
00019 using Markov::API::CUDA::CUDADeviceController;
00020
00021 int main(int argc, char** argv) {
00022
00023
00024
00025     Markov::API::CUDA::CUDAModelMatrix markovPass;
00026     std::cerr « "Importing model.\n";
00027     markovPass.Import("models/finished.mdl");
00028     markovPass.Buff("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~", 10, true, true);
00029     std::cerr « "Import done. \n";
00030     markovPass.ConstructMatrix();
00031     //markovPass.DumpJSON();
00032     CUDADeviceController::ListCudaDevices();
00033
00034     std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
00035     std::cerr « "Starting walk. \n";
00036     markovPass.FastRandomWalk(1000000000,"/media/ignis/Stuff/wordlist.txt",12,12, false, false);
00037     //markovPass.FastRandomWalk(500000000,"/media/ignis/Stuff/wordlist2.txt",6,12,25, true);
00038     std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
00039
00040     std::cerr « "Finished in:" « std::chrono::duration_cast<std::chrono::milliseconds> (end -
       begin).count() « " milliseconds" « std::endl;
00041
00042
00043 }
```

9.19 Markopy/documentation/dirs.docs File Reference

doxygen information about the directories

9.19.1 Detailed Description

doxygen information about the directories
Definition in file dirs.docs.

9.20 dirs.docs

```
00001
00002 /**
00003  * @file dirs.docs
00004  * @brief doxygen information about the directories
00005  */
00006
00007 /**
00008  * @dir Markopy/Markopy
```

```
00009  * @brief CPython extension for MarkovAPI
00010  */
00011
00012 /**
00013  * @dir Markopy/Markopy/src/CLI
00014  * @brief Python CLI for Markopy
00015  */
00016
00017 /**
00018  * @dir Markopy/Markopy/src/Module
00019  * @brief CPP module for Markopy
00020  */
00021
00022 /**
00023  * @dir Markopy/CudaMarkopy
00024  * @brief CPython extension for CudaMarkovAPI
00025  */
00026
00027 /**
00028  * @dir Markopy/CudaMarkopy/src/CLI
00029  * @brief Python CLI for CudaMarkopy
00030  */
00031
00032 /**
00033  * @dir Markopy/CudaMarkopy/src/Module
00034  * @brief CPP module for CudaMarkopy
00035  */
00036
00037 /**
00038  * @dir Markopy/MarkovAPI
00039  * @brief Base project
00040  */
00041
00042 /**
00043  * @dir Markopy/CudaMarkovAPI
00044  * @brief Markov API with CUDA Acceleration
00045  */
00046
00047 /**
00048  * @dir Markopy/MarkovAPICLI
00049  * @brief Test utility for MarkovAPI
00050  */
00051
00052 /**
00053  * @dir Markopy/MarkovModel
00054  * @brief Header only markov library
00055  */
00056
00057 /**
00058  * @dir Markopy/MarkovPasswordsGUI
00059  * @brief Graphical Interface for MarkovAPI
00060  */
00061
00062 /**
00063  * @dir Markopy/UnitTests
00064  * @brief Unit tests for MarkovAPI
00065  */
00066
00067 /**
00068  * @dir Markopy/documentation
00069  * @brief Files related to documentation generation
00070  */
```

9.21 Markopy/Markopy/src/CLI/base.py File Reference

base command line interface for python

Classes

- class Python.Markopy.BaseCLI

  Base CLI class to handle user interactions

- class Python.Markopy.AbstractGenerationModelCLI

  abstract class for generation capable models

- class Python.Markopy.AbstractTrainingModelCLI

  abstract class for training capable models

Namespaces

- base

9.21.1 Detailed Description

base command line interface for python
Definition in file base.py.

9.22 base.py

```
00001 #!/usr/bin/python3
00002
00003
00007
00008 import argparse
00009 import allogate as logging
00010 import os
00011 from abc import abstractmethod
00012 from termcolor import colored
00013 from mm import MarkovModel
00014
00015
00016 class BaseCLI():
00017     """! @brief Base CLI class to handle user interactions
00018          @belongsto Python::Markopy
00019     """
00020     def __init__(self, add_help : bool=True):
00021         """!
00022         @brief initialize base CLI
00023         @param add_help decide to overload the help function or not
00024         """
00025         self.parserparser = argparse.ArgumentParser(description="Python wrapper for MarkovPasswords.",
00026         epilog=f"""{colored("Sample runs:", "yellow")}
00027         {__file__.split("/")[-1]} train untrained.mdl -d dataset.dat -s "\\t" -o trained.mdl
00028             Import untrained.mdl, train it with dataset.dat which has tab delimited data, output
      resulting model to trained.mdl\n
00029
00030         {__file__.split("/")[-1]} generate trained.mdl -n 500 -w output.txt
00031             Import trained.mdl, and generate 500 lines to output.txt
00032
00033         {__file__.split("/")[-1]} combine untrained.mdl -d dataset.dat -s "\\t" -n 500 -w output.txt
00034             Train and immediately generate 500 lines to output.txt. Do not export trained model.
00035
00036         {__file__.split("/")[-1]} combine untrained.mdl -d dataset.dat -s "\\t" -n 500 -w output.txt
      -o trained.mdl
00037             Train and immediately generate 500 lines to output.txt. Export trained model.
00038         """, add_help=add_help, formatter_class=argparse.RawTextHelpFormatter)
00039         self.print_helpprint_help = self.parserparser.print_help
00040         self.modelmodel = MarkovModel()
00041
00042     @abstractmethod
00043     def add_arguments(self):
00044         "! @brief Add command line arguements to the parser"
00045         self.parserparser.add_argument("mode",                              help="Process mode. Either
      'Train', 'Generate', or 'Combine'.")
00046         self.parserparser.add_argument("-t", "--threads",default=10,        help="Number of lines to
      generate. Ignored in training mode.")
00047         self.parserparser.add_argument("-v", "--verbosity",action="count", help="Output verbosity.")
00048         self.parserparser.add_argument("-b", "--bulk",action="store_true", help="Bulk generate or bulk
      train every corpus/model in the folder.")
00049
00050     @abstractmethod
00051     def help(self):
00052         "! @brief Handle help strings. Defaults to argparse's help"
00053         self.print_helpprint_help()
00054
00055     def parse(self):
00056         "! @brief add, parse and hook arguements"
00057         self.add_argumentsadd_arguments()
00058         self.parse_argumentsparse_arguments()
00059         self.init_post_argumentsinit_post_arguments()
00060
00061     @abstractmethod
00062     def init_post_arguments(self):
00063         "! @brief set up stuff that is collected from command line arguements"
00064         logging.VERBOSITY = 0
00065         try:
00066             if self.argsargs.verbosity:
00067                 logging.VERBOSITY = self.argsargs.verbosity
00068                 logging.pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00069         except:
```

```
00070             pass
00071
00072     @abstractmethod
00073     def parse_arguments(self):
00074         "! @brief trigger parser"
00075         self.argsargs = self.parserparser.parse_known_args()[0]
00076
00077     def import_model(self, filename : str):
00078         """!
00079         @brief Import a model file
00080         @param filename filename to import
00081         """
00082         logging.pprint("Importing model file.", 1)
00083
00084         if not self.check_import_pathcheck_import_path(filename):
00085             logging.pprint(f"Model file at {filename} not found. Check the file path, or working
      directory")
00086             return False
00087
00088         self.modelmodel.Import(filename)
00089         logging.pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
      bool=False):
00095         """!
00096             @brief Train a model via CLI parameters
00097             @param model Model instance
00098             @param dataset filename for the dataset
00099             @param seperator seperator used with the dataset
00100             @param output output filename
00101             @param output_forced force overwrite
00102             @param bulk marks bulk operation with directories
00103         """
00104         logging.pprint("Training.")
00105
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging.pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced
      else"} and -s/--seperator parameters. Exiting.")
00108             return False
00109
00110         if not bulk and not self.check_corpus_pathcheck_corpus_path(dataset):
00111             logging.pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113
00114         if not self.check_export_pathcheck_export_path(output):
00115             logging.pprint(f"Cannot create output at {output}")
00116             return False
00117
00118         if(seperator == '\\t'):
00119             logging.pprint("Escaping seperator.", 3)
00120             seperator = '\t'
00121
00122         if(len(seperator)!=1):
00123             logging.pprint(f'Delimiter must be a single character, and "{seperator}" is not
      accepted.')
00124             exit(4)
00125
00126         logging.pprint(f'Starting training.', 3)
00127         self.modelmodel.Train(dataset,seperator, int(self.argsargs.threads))
00128         logging.pprint(f'Training completed.', 2)
00129
00130         if(output):
00131             logging.pprint(f'Exporting model to {output}', 2)
00132             self.exportexport(output)
00133         else:
00134             logging.pprint(f'Model will not be exported.', 1)
00135
00136         return True
00137
00138     def export(self, filename : str):
00139         """!
00140         @brief Export model to a file
00141         @param filename filename to export to
00142         """
00143         self.modelmodel.Export(filename)
00144
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """!
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.argsargs.count):
```

```
00153             logging.pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
      Exiting.")
00154             return False
00155 
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging.pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate_generate(wordlist)
00159 
00160     @abstractmethod
00161     def _generate(self, wordlist : str):
00162         """!
00163         @brief wrapper for generate function. This can be overloaded by other models
00164         @param wordlist filename to generate to
00165         """
00166         self.modelmodel.Generate(int(self.argsargs.count), wordlist, int(self.argsargs.min),
      int(self.argsargs.max), int(self.argsargs.threads))
00167 
00168     @staticmethod
00169     def check_import_path(filename : str):
00170         """!
00171         @brief check import path for validity
00172         @param filename filename to check
00173         """
00174 
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179 
00180     @staticmethod
00181     def check_corpus_path(filename : str):
00182         """!
00183         @brief check import path for validity
00184         @param filename filename to check
00185         """
00186 
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
00191     @staticmethod
00192     def check_export_path(filename : str):
00193         """!
00194         @brief check import path for validity
00195         @param filename filename to check
00196         """
00197 
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201 
00202     def process(self):
00203         """!
00204         @brief Process parameters for operation
00205         """
00206         if(self.argsargs.bulk):
00207             logging.pprint(f"Bulk mode operation chosen.", 4)
00208             if (self.argsargs.mode.lower() == "train"):
00209                 if (os.path.isdir(self.argsargs.output) and not os.path.isfile(self.argsargs.output))
      and (os.path.isdir(self.argsargs.dataset) and not os.path.isfile(self.argsargs.dataset)):
00210                     corpus_list = os.listdir(self.argsargs.dataset)
00211                     for corpus in corpus_list:
00212                         self.import_modelimport_model(self.argsargs.input)
00213                         logging.pprint(f"Training {self.args.input} with {corpus}", 2)
00214                         output_file_name = corpus
00215                         model_extension = ""
00216                         if "." in self.argsargs.input:
00217                             model_extension = self.argsargs.input.split(".")[-1]
00218                         self.traintrain(f"{self.args.dataset}/{corpus}", self.argsargs.seperator,
      f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00219                 else:
00220                     logging.pprint("In bulk training, output and dataset should be a directory.")
00221                     exit(1)
00222 
00223             elif (self.argsargs.mode.lower() == "generate"):
00224                 if (os.path.isdir(self.argsargs.wordlist) and not
      os.path.isfile(self.argsargs.wordlist)) and (os.path.isdir(self.argsargs.input) and not
      os.path.isfile(self.argsargs.input)):
00225                     model_list = os.listdir(self.argsargs.input)
00226                     print(model_list)
00227                     for input in model_list:
00228                         logging.pprint(f"Generating from {self.args.input}/{input} to
      {self.args.wordlist}/{input}.txt", 2)
00229                         self.import_modelimport_model(f"{self.args.input}/{input}")
00230                         model_base = input
00231                         if "." in self.argsargs.input:
00232                             model_base = input.split(".")[1]
```

```
00233                         self.generategenerate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00234                 else:
00235                     logging.pprint("In bulk generation, input and wordlist should be directory.")
00236 
00237         else:
00238             self.import_modelimport_model(self.argsargs.input)
00239             if (self.argsargs.mode.lower() == "generate"):
00240                 self.generategenerate(self.argsargs.wordlist)
00241 
00242 
00243             elif (self.argsargs.mode.lower() == "train"):
00244                 self.traintrain(self.argsargs.dataset, self.argsargs.seperator, self.argsargs.output,
      output_forced=True)
00245 
00246 
00247             elif(self.argsargs.mode.lower() == "combine"):
00248                 self.traintrain(self.argsargs.dataset, self.argsargs.seperator, self.argsargs.output)
00249                 self.generategenerate(self.argsargs.wordlist)
00250 
00251 
00252             else:
00253                 logging.pprint("Invalid mode arguement given.")
00254                 logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255                 exit(5)
00256 
00257 class AbstractGenerationModelCLI(BaseCLI):
00258     """!
00259     @brief abstract class for generation capable models
00260     @belongsto Python::Markopy
00261     @extends Python::Markopy::BaseCLI
00262     """
00263     @abstractmethod
00264     def add_arguments(self):
00265         "Add command line arguements to the parser"
00266         super().add_arguments()
00267         self.parserparser.add_argument("input",                              help="Input model file.
      This model will be imported before starting operation.")
00268         self.parserparser.add_argument("-w", "--wordlist",                   help="Wordlist file path to
      export generation results to. Will be ignored for training mode")
00269         self.parserparser.add_argument("--min", default=6,                   help="Minimum length that
      is allowed during generation")
00270         self.parserparser.add_argument("--max", default=12,                  help="Maximum length that
      is allowed during generation")
00271         self.parserparser.add_argument("-n", "--count",                      help="Number of lines to
      generate. Ignored in training mode.")
00272 
00273 
00274 class AbstractTrainingModelCLI(AbstractGenerationModelCLI, BaseCLI):
00275     """!
00276     @brief abstract class for training capable models
00277     @belongsto Python::Markopy
00278     @extends Python::Markopy::BaseCLI
00279     @extends Python::Markopy::AbstractGenerationModelCLI
00280     """
00281     @abstractmethod
00282     def add_arguments(self):
00283         "Add command line arguements to the parser"
00284         self.parserparser.add_argument("-o", "--output",                     help="Output model file.
      This model will be exported when done. Will be ignored for generation mode.")
00285         self.parserparser.add_argument("-d", "--dataset",                    help="Dataset file to read
      input from for training. Will be ignored for generation mode.")
00286         self.parserparser.add_argument("-s", "--seperator",                  help="Seperator character
      to use with training data.(character between occurrence and value)")
00287         super().add_arguments()
```

9.23 Markopy/Markopy/src/CLI/evaluate.py File Reference

Evaluation of model integrity and score.

Classes

- class Python.Markopy.Evaluation.Evaluator

  Abstract class to evaluate and score integrity/validty.

- class Python.Markopy.Evaluation.ModelEvaluator

  evaluate a model

- class Python.Markopy.Evaluation.CorpusEvaluator

  evaluate a corpus

Namespaces

- evaluate
- Python.Markopy.Evaluation

  namespace for integrity evaluation

9.23.1 Detailed Description

Evaluation of model integrity and score.
Definition in file evaluate.py.

9.24 evaluate.py

```
00001
00002
00003
00004
00008 from abc import abstractmethod
00009 import re
00010 import allogate as logging
00011 import statistics
00012 import os
00013 from copy import copy
00014 import glob
00015 import re
00016
00017
00021
00022 class Evaluator:
00023     """!
00024     @brief Abstract class to evaluate and score integrity/validty
00025     @belongsto Python::Markopy::Evaluation
00026     """
00027     def __init__(self, filename: str) -> None:
00028         """!
00029         @brief default constructor for evaluator
00030         @param filename filename to evaluate. Can be a pattern
00031         """
00032         self.filenamefilename = filename
00033         self.checkschecks = []
00034         self.TEST_PASS_SYMBOLTEST_PASS_SYMBOL = b"\xe2\x9c\x85".decode()
00035         self.TEST_FAIL_SYMBOLTEST_FAIL_SYMBOL = b"\xe2\x9d\x8c".decode()
00036         self.all_checks_passedall_checks_passed = True
00037         self.filesfiles = []
00038         if("*" in filename):
00039             self.filesfiles = glob.glob(filename)
00040         else:
00041             self.filesfiles.append(filename)
00042         return True
00043
00044     def evaluate(self) -> bool:
00045         "! @brief base evaluation function"
00046         for file in self.filesfiles:
00047             self._evaluate_evaluate(file)
00048
00049         self.check_funcscheck_funcs = [func for func in dir(self) if (callable(getattr(self, func))
      and func.startswith("check_"))]
00050
00051     @abstractmethod
00052     def _evaluate(self, file) -> list:
00053         """!
00054         @brief internal evaluation function for a single file
00055         @param file filename to evaluate
00056         """
00057         if(not os.path.isfile(file)):
00058             logging.pprint(f"Given file {file} is not a valid filename")
00059             return False
00060         else:
00061              return open(file, "rb").read().split(b"\n")
00062
00063
00064
00065     def success(self, checkname):
00066         """!
00067         @brief pass a test
00068         @param checkname text to display with the check
00069         """
00070         self.checkschecks.append((checkname, self.TEST_PASS_SYMBOLTEST_PASS_SYMBOL))
00071
00072     def fail(self, checkname):
```

```
00073         """!
00074         @brief fail a test
00075         @param checkname text to display with the check
00076         """
00077
00078         self.all_checks_passedall_checks_passed = False
00079         self.checkschecks.append((checkname, self.TEST_FAIL_SYMBOLTEST_FAIL_SYMBOL))
00080
00081     def finalize(self):
00082         "! @brief finalize an evaluation and print checks"
00083         print("\n################ Checks ################ ")
00084         for test in self.checkschecks:
00085             logging.pprint(f"{test[0]:30}:{test[1]} ")
00086         print("\n")
00087         self.checkschecks = []
00088         return self.all_checks_passedall_checks_passed
00089
00090 class ModelEvaluator(Evaluator):
00091     """!
00092     @brief evaluate a model
00093     @belongsto Python::Markopy::Evaluation
00094     @extends Python::Markopy::Evaluation::Evaluator
00095     """
00096     def __init__(self, filename: str) -> None:
00097         "! @brief default constructor"
00098         valid = super().__init__(filename)
00099
00100         if not valid:
00101             return False
00102
00103     def evaluate(self):
00104         "! @brief evaluate a model"
00105         logging.VERBOSITY=2
00106         logging.SHOW_STACK_THRESHOLD=3
00107         super().evaluate()
00108         for file in self.filesfiles:
00109             logging.pprint(f"Model: {file.split('/')[-1]}: ",2)
00110             edges = super()._evaluate(file)
00111             if not edges:
00112                 continue
00113             self.lnodeslnodes = {}
00114             self.rnodesrnodes = {}
00115             self.ewsews = []
00116             self.edge_countedge_count = len(edges)
00117             for edge in edges:
00118                 if(edge ==b''):
00119                     self.edge_countedge_count-=1
00120                     continue
00121                 try:
00122                     e = edge.split(b',')
00123                     self.ewsews.append(int(edge[2:-2:1]))
00124                     if(e[0] not in self.lnodeslnodes):
00125                         self.lnodeslnodes[e[0]]=1
00126                     else:
00127                         self.lnodeslnodes[e[0]]+=1
00128                     if(e[-1] not in self.rnodesrnodes):
00129                         self.rnodesrnodes[e[-1]]=1
00130                     else:
00131                         self.rnodesrnodes[e[-1]]+=1
00132                 except Exception as e:
00133                     print(e)
00134                     logging.pprint(f"Model file is corrupted.", 0)
00135                     continue
00136
00137             self.lnode_countlnode_count = len(self.lnodeslnodes)
00138             self.rnode_countrnode_count = len(self.rnodesrnodes)
00139             logging.pprint(f"total edges: {self.edge_count}", 1)
00140             logging.pprint(f"unique left nodes: {self.lnode_count}", 1)
00141             logging.pprint(f"unique right nodes: {self.rnode_count}", 1)
00142
00143             for check in self.check_funcscheck_funcs:
00144                 try:
00145                     self.__getattribute__(check)()
00146                 except Exception as e:
00147                     print(e)
00148                     self.failfail(f"Exceptionn in {check}")
00149             self.finalizefinalize()
00150
00151     def check_dangling(self):
00152         "! @brief check if model has dangling nodes"
00153         if(self.lnode_countlnode_count == self.rnode_countrnode_count):
00154             self.successsuccess("No dangling nodes")
00155         else:
00156             logging.pprint(f"Dangling nodes found, lnodes and rnodes do not match", 0)
00157             self.failfail("No dangling nodes")
00158
00159     def check_structure(self):
```

```
00160         "! @brief check model structure for validity"
00161         if((self.lnode_countlnode_count-1) * (self.rnode_countrnode_count-1) +
    2*(self.lnode_countlnode_count-1)):
00162             self.successsuccess("Model structure")
00163         else:
00164             logging.pprint(f"Model did not satisfy structural integrity check (lnode_count-1) *
    (rnode_count-1) + 2*(lnode_count-1)", 0)
00165             self.failfail("Model structure")
00166 
00167     def check_weight_deviation(self):
00168         "! @brief check model standart deviation between edge weights"
00169         mean = sum(self.ewsews) / len(self.ewsews)
00170         variance = sum([(x - mean) ** 2) for x in self.ewsews]) / len(self.ewsews)
00171         res = variance ** 0.5
00172         self.stdevstdev = res
00173         if(res==0):
00174             logging.pprint(f"Model seems to be untrained", 0)
00175             self.failfail("Model has any training")
00176         else:
00177             self.successsuccess("Model has any training")
00178         if(res<3000):
00179             logging.pprint(f"Model is not adequately trained. Might result in inadequate results", 1)
00180             self.failfail("Model has training")
00181             self.failfail(f"Model training score: {round(self.stdev,2)}")
00182         else:
00183             self.successsuccess("Model has training")
00184             self.successsuccess(f"Model training score: {round(self.stdev)}")
00185 
00186     def check_min(self):
00187         "! @brief check 0 edge weights distribution"
00188         count = 0
00189         for ew in self.ewsews:
00190             if ew==0:
00191                 count+=1
00192         if(count > self.rnode_countrnode_count*0.8):
00193             self.failfail("Too many 0 edges")
00194             logging.pprint(f"0 weighted edges are dangerous and may halt the model.", 0)
00195         else:
00196             self.successsuccess("0 edges below threshold")
00197 
00198     def check_min_10percent(self):
00199         "! @brief check minimum 10% of the edges"
00200         sample = self.ewsews[int(self.edge_countedge_count*0.1)]
00201         #print(f"10per: {sample}")
00202         avg = sum(self.ewsews) / len(self.ewsews)
00203         #print(f"avg: {avg}")
00204         med = statistics.median(self.ewsews)
00205         #print(f"med: {med}")
00206 
00207     def check_lean(self):
00208         "! @brief check which way model is leaning. Left, or right"
00209         sample = self.ewsews[int(self.edge_countedge_count*0.1)]
00210         avg = sum(self.ewsews) / len(self.ewsews)
00211         med = statistics.median(self.ewsews)
00212 
00213         if(med*10<sample):
00214             logging.pprint("Median is too left leaning and might indicate high entropy")
00215             self.failfail("Median too left leaning")
00216         else:
00217             self.successsuccess("Median in expected ratio")
00218         pass
00219 
00220         if(sample*5>avg):
00221             logging.pprint("Least probable 10% too close to average, might indicate inadequate
    training")
00222             self.failfail("Bad bottom 10%")
00223         else:
00224             self.successsuccess("Good bottom 10%")
00225         pass
00226 
00227 
00228     def check_distrib(self):
00229         "! @deprecated"
00230         sorted_ews = copy(self.ewsews)
00231         sorted_ews.sort(reverse=True)
00232         ratio1 = sorted_ews[0]/sorted_ews[int(self.edge_countedge_count/2)]
00233         ratio2 =
    sorted_ews[int(self.edge_countedge_count/2)]/sorted_ews[int(self.edge_countedge_count*0.1)]
00234         #print(ratio1)
00235         #print(ratio2)
00236 
00237 
00238 class CorpusEvaluator(Evaluator):
00239     """!
00240     @brief evaluate a corpus
00241     @belongsto Python::Markopy::Evaluation
00242     @extends Python::Markopy::Evaluation::Evaluator
```

```
00243     """
00244     def __init__(self, filename: str) -> None:
00245         """!
00246         @brief default constructor
00247         @param filename filename or pattern to check
00248         """
00249         valid = super().__init__(filename)
00250         if not valid:
00251             return False
00252 
00253     def evaluate(self):
00254         "! @brief evalute a corpus. Might take a long time"
00255         logging.pprint("WARNING: This takes a while with larger corpus files", 2)
00256         logging.VERBOSITY=2
00257         logging.SHOW_STACK_THRESHOLD=3
00258         super().evaluate()
00259         for file in self.filesfiles:
00260 
00261             delimiter = ”
00262             sum=0
00263             max=0
00264             total_chars = 0
00265             lines_count = 0
00266             bDelimiterConflict=False
00267             logging.pprint(f"Corpus: {file.split(’/’)[-1]}: ",2)
00268             with open(file, "rb") as corpus:
00269                 for line in corpus:
00270                     lines_count+=1
00271                     match = re.match(r"([0-9]+)(.)(.*)\n", line.decode()).groups()
00272                     if(delimiter and delimiter!=match[1]):
00273                         bDelimiterConflict = True
00274 
00275                     elif(not delimiter):
00276                         delimiter = match[1]
00277                         logging.pprint(f"Delimiter is: {delimiter.encode()}")
00278                     sum +=int(match[0])
00279                     total_chars += len(match[2])
00280                     if(int(match[0])>max):
00281                         max=int(match[0])
00282 
00283                 if(bDelimiterConflict):
00284                     self.failfail("Incorrect delimiter found")
00285                 else:
00286                     self.successsuccess("No structural conflicts")
00287 
00288                 logging.pprint(f"Total number of lines: {lines_count}")
00289                 logging.pprint(f"Sum of all string weights: {sum}")
00290                 logging.pprint(f"Character total: {total_chars}")
00291                 logging.pprint(f"Average length: {total_chars/lines_count}")
00292                 logging.pprint(f"Average weight: {sum/lines_count}")
00293 
00294             self.finalizefinalize()
00295     def _evaluate(self, file) -> list:
00296         """!
00297         @brief evaluate a single file. Remove reading file because it should be read line by line.
00298         @param file corpus filename to evaluate
00299         """
00300         if(not os.path.isfile(file)):
00301             logging.pprint(f"Given file {file} is not a valid filename")
00302             return False
00303         else:
00304             return True
00305 
```

9.25 Markopy/Markopy/src/CLI/importer.py File Reference

dynamic import wrapper for markopy model

Namespaces

- importer

Functions

- def importer.import_markopy ()

  import and return markopy module

9.25.1 Detailed Description

dynamic import wrapper for markopy model
Definition in file importer.py.

9.26 importer.py

```
00001
00002
00006
00007 from importlib.util import spec_from_loader, module_from_spec
00008 from importlib.machinery import SourceFileLoader, ExtensionFileLoader
00009 import os
00010
00011
00012 def import_markopy():
00013     """! @brief import and return markopy module
00014          @returns markopy module
00015     """
00016     ext = "so"
00017     if os.name == 'nt':
00018         ext="pyd"
00019     try:
00020         spec = spec_from_loader("markopy", ExtensionFileLoader("markopy",
       os.path.abspath(f"markopy.{ext}")))
00021         markopy = module_from_spec(spec)
00022         spec.loader.exec_module(markopy)
00023         return markopy
00024     except ImportError as e:
00025         #print(f"({__file__}) Working in development mode. Trying to load markopy.{ext} from
       ../../../out/")
00026         if(os.path.exists(f"../../../out/lib/markopy.{ext}")):
00027             spec = spec_from_loader("markopy", ExtensionFileLoader("markopy",
       os.path.abspath(f"../../../out/lib/markopy.{ext}")))
00028             markopy = module_from_spec(spec)
00029             spec.loader.exec_module(markopy)
00030             return markopy
00031         else:
00032             raise e
```

9.27 Markopy/Markopy/src/CLI/markopy.py File Reference

Entry point for markopy scripts.

Classes

- class Python.Markopy.MarkopyCLI

  Top level model selector for Markopy CLI.

Namespaces

- markopy

Variables

- string markopy.ext = "so"
- markopy.markopy = import_markopy()
- markopy.mp = MarkopyCLI()

9.27.1 Detailed Description

Entry point for markopy scripts.
Definition in file markopy.py.

9.28 markopy.py

```
00001 #!/usr/bin/env python3
00002
00003
00004
00008
00009
00013
00014
00018
00019 from importlib.util import spec_from_loader, module_from_spec
00020 from importlib.machinery import SourceFileLoader, ExtensionFileLoader
00021 import os
00022 import sys
00023
00024 ext = "so"
00025 if os.name == 'nt':
00026     ext="pyd"
00027
00028
00029 try:
00030     from base import BaseCLI
00031     from mp import MarkovPasswordsCLI
00032     from mmx import ModelMatrixCLI
00033     from evaluate import CorpusEvaluator, ModelEvaluator
00034     from importer import import_markopy
00035     markopy = import_markopy()
00036
00037
00038 except ModuleNotFoundError as e:
00039     #print("Working in development mode. Trying to load markopy.py from ../../../Markopy/")
00040     if(os.path.exists("../../../Markopy/src/CLI/base.py")):
00041         sys.path.insert(1, '../../../Markopy/src/CLI/')
00042         from base import BaseCLI
00043         from mp import MarkovPasswordsCLI
00044         from mmx import ModelMatrixCLI
00045         from evaluate import CorpusEvaluator, ModelEvaluator
00046         from importer import import_markopy
00047         markopy = import_markopy()
00048
00049     else:
00050         raise e
00051
00052
00053 from termcolor import colored
00054 from abc import abstractmethod
00055
00056 class MarkopyCLI(BaseCLI):
00057     """!
00058         @brief Top level model selector for Markopy CLI.
00059         This class is used for injecting the -mt parameter to the CLI, and determining the model type
      depending on that.
00060         @belongsto Python::Markopy
00061         @extends Python::Markopy::BaseCLI
00062         @extends Python::Markopy::ModelMatrixCLI
00063         @extends Python::Markopy::MarkovPasswordsCLI
00064     """
00065
00066     def __init__(self, add_help=False):
00067         """!
00068         @brief default constructor
00069         """
00070
00071         BaseCLI.__init__(self,add_help)
00072         self.argsargsargs = None
00073         self.parserparser.epilog = f"""
00074         {colored("Sample runs:", "yellow")}
00075         {__file__.split("/")[-1]} -mt MP generate trained.mdl -n 500 -w output.txt
00076             Import trained.mdl, and generate 500 lines to output.txt
00077
00078         {__file__.split("/")[-1]} -mt MMX generate trained.mdl -n 500 -w output.txt
00079             Import trained.mdl, and generate 500 lines to output.txt
00080         """
00081
00082     @abstractmethod
00083     def add_arguments(self):
00084         """!
00085         @brief add -mt/--model_type constructor
00086         """
00087         self.parserparser.add_argument("-mt", "--model_type", default="_MMX", help="Model type to use.
      Accepted values: MP, MMX")
00088         self.parserparser.add_argument("-h", "--help", action="store_true", help="Model type to use.
      Accepted values: MP, MMX")
00089         self.parserparser.add_argument("-ev", "--evaluate", help="Evaluate a models integrity")
00090         self.parserparser.add_argument("-evt", "--evaluate_type", help="Evaluation type, model or
      corpus")
```

```
00091         self.parserparser.print_help = self.helphelphelp
00092
00093     @abstractmethod
00094     def help(self):
00095         """!
00096         @brief overload help function to print submodel helps
00097         """
00098         self.parserparser.print_help = self.stubstub
00099         self.argsargsargs = self.parserparser.parse_known_args()[0]
00100         if(self.argsargsargs.model_type!="_MMX"):
00101             if(self.argsargsargs.model_type=="MP"):
00102                 mp = MarkovPasswordsCLI()
00103                 mp.add_arguments()
00104                 mp.parser.print_help()
00105             elif(self.argsargsargs.model_type=="MMX"):
00106                 mp = ModelMatrixCLI()
00107                 mp.add_arguments()
00108                 mp.parser.print_help()
00109         else:
00110             print(colored("Model Mode selection choices:", "green"))
00111             self.print_helpprint_help()
00112             print(colored("Following are applicable for -mt MP mode:", "green"))
00113             mp = MarkovPasswordsCLI()
00114             mp.add_arguments()
00115             mp.parser.print_help()
00116             print(colored("Following are applicable for -mt MMX mode:", "green"))
00117             mp = ModelMatrixCLI()
00118             mp.add_arguments()
00119             mp.parser.print_help()
00120
00121         exit()
00122
00123
00124     @abstractmethod
00125     def parse(self):
00126         "! @brief overload parse function to parse for submodels"
00127
00128         self.add_argumentsadd_argumentsadd_argumentsadd_argumentsadd_argumentsadd_arguments()
00129         self.parse_argumentsparse_arguments()
00130
      self.init_post_argumentsinit_post_argumentsinit_post_argumentsinit_post_argumentsinit_post_arguments()
00131         if(self.argsargsargs.evaluate):
00132             self.evaluateevaluate(self.argsargsargs.evaluate)
00133             exit()
00134         if(self.argsargsargs.model_type == "MP"):
00135             self.clicli = MarkovPasswordsCLI()
00136         elif(self.argsargsargs.model_type == "MMX" or self.argsargsargs.model_type == "_MMX"):
00137             self.clicli = ModelMatrixCLI()
00138         else:
00139             self.parse_failparse_fail()
00140
00141         if(self.argsargsargs.help): return self.helphelphelp()
00142         self.clicli.parse()
00143
00144     @abstractmethod
00145     def init_post_arguments(self):
00146         pass
00147
00148     @abstractmethod
00149     def parse_fail(self):
00150         "! @brief failed to parse model type"
00151         print("Unrecognized model type.")
00152         exit()
00153
00154     def process(self):
00155         "! @brief pass the process request to selected submodel"
00156         return self.clicli.process()
00157
00158     def stub(self):
00159         "! @brief stub function to hack help requests"
00160         return
00161
00162
00163     def evaluate(self,filename : str):
00164         if(not self.args.evaluate_type):
00165             if(filename.endswith(".mdl")):
00166                 ModelEvaluator(filename).evaluate()
00167             else:
00168                 CorpusEvaluator(filename).evaluate()
00169         else:
00170             if(self.args.evaluate_type == "model"):
00171                 ModelEvaluator(filename).evaluate()
00172             else:
00173                 CorpusEvaluator(filename).evaluate()
00174
00175     def init_post_arguments(sel):
00176         pass
```

```
00177
00178 if __name__ == "__main__":
00179     mp = MarkopyCLI()
00180     mp.parse()
00181     mp.process()
```

9.29 Markopy/Markopy/src/CLI/mm.py File Reference

Abstract representation of CPP/Python intermediate layer classes.

Classes

- class Python.Markopy.MarkovModel

  Abstract representation of a markov model.
- class Python.Markopy.ModelMatrix

  Abstract representation of a matrix based model.

Namespaces

- mm

Variables

- mm.markopy = import_markopy()

9.29.1 Detailed Description

Abstract representation of CPP/Python intermediate layer classes.
Definition in file mm.py.

9.30 mm.py

```
00001
00002
00003
00007
00008 from abc import abstractmethod
00009
00010 from importer import import_markopy
00011 markopy = import_markopy()
00012
00013 class MarkovModel(markopy.MarkovPasswords):
00014     """!
00015     @brief Abstract representation of a markov model
00016     @implements Markov::API::MarkovPasswords
00017     @belongsto Python::Markopy
00018
00019     To help with the python-cpp gateway documentation.
00020     """
00021     @abstractmethod
00022     def Import(filename : str):
00023         pass
00024
00025     @abstractmethod
00026     def Export(filename : str):
00027         pass
00028
00029     @abstractmethod
00030     def Train(dataset: str, seperator : str, threads : int):
00031         pass
00032
00033     @abstractmethod
00034     def Generate(count : int, wordlist : str, minlen : int, maxlen: int, threads : int):
00035         pass
00036
00037
00038 class ModelMatrix(markopy.ModelMatrix):
00039     """!
00040     @brief Abstract representation of a matrix based model
00041     @implements Markov::API::ModelMatrix
```

```
00042     @belongsto Python::Markopy
00043
00044     To help with the python-cpp gateway documentation.
00045     """
00046
00047     @abstractmethod
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass
```

9.31 Markopy/Markopy/src/CLI/mmx.py File Reference

ModelMatrix CLI wrapper.

Classes

- class Python.Markopy.ModelMatrixCLI

  Extension of Python.Markopy.Base.BaseCLI for Markov::API::ModelMatrix.

Namespaces

- mmx

Variables

- mmx.markopy = import_markopy()
- mmx.mp = ModelMatrixCLI()

9.31.1 Detailed Description

ModelMatrix CLI wrapper.
Definition in file mmx.py.

9.32 mmx.py

```
00001 #!/usr/bin/python3
00002
00003
00007
00008
00009 from mm import ModelMatrix
00010
00011 from importer import import_markopy
00012 markopy = import_markopy()
00013
00014 from base import BaseCLI, AbstractGenerationModelCLI
00015 import os
00016 import allogate as logging
00017
00018 class ModelMatrixCLI(AbstractGenerationModelCLI, ModelMatrix):
00019     """!
00020         @brief Extension of Python.Markopy.Base.BaseCLI  for Markov::API::ModelMatrix
00021         @belongsto Python::Markopy
00022         @extends Python::Markopy::AbstractGenerationModelCLI
00023         @extends Python::Markopy::ModelMatrix
00024
00025         adds -st/--stdout arguement to the command line.
00026     """
00027     def __init__(self, add_help:bool=True):
00028         "! @brief initialize model with Markov::API::ModelMatrix"
00029         super().__init__(add_help)
00030         self.modelmodelmodel = markopy.ModelMatrix()
00031
00032     def add_arguments(self):
00033         super().add_arguments()
00034         self.parserparser.add_argument("-st", "--stdout", action="store_true", help="Stdout mode")
00035
00036     def init_post_arguments(self):
00037         super().init_post_arguments()
00038         self.fileIOfileIO = not self.argsargs.stdout
00039
00040     def _generate(self, wordlist : str, ):
```

```
00041         self.modelmodelmodel.FastRandomWalk(int(self.argsargs.count), wordlist,
      int(self.argsargs.min), int(self.argsargs.max), int(self.argsargs.threads), self.fileIOfileIO)
00042
00043 if __name__ == "__main__":
00044     mp = ModelMatrixCLI()
00045     mp.parse()
00046     mp.process()
```

9.33 Markopy/Markopy/src/CLI/mp.py File Reference

CLI wrapper for MarkovPasswords.

Classes

- class Python.Markopy.MarkovPasswordsCLI

  Extension of Python.Markopy.Base.BaseCLI for Markov::API::MarkovPasswords.

Namespaces

- mp

Variables

- mp.markopy = import_markopy()
- mp.mp = MarkovPasswordsCLI()

9.33.1 Detailed Description

CLI wrapper for MarkovPasswords.
Definition in file mp.py.

9.34 mp.py

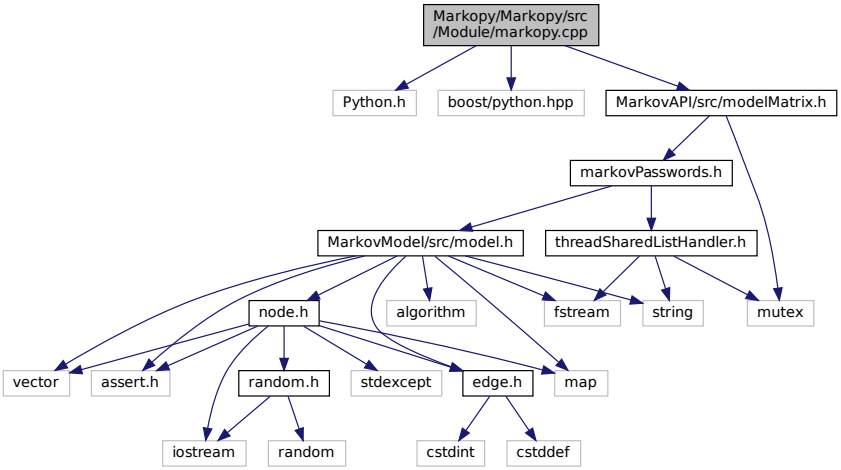
```
00001 #!/usr/bin/python3
00002
00003
00007
00008 from importlib.util import spec_from_loader, module_from_spec
00009 from importlib.machinery import SourceFileLoader, ExtensionFileLoader
00010 import os
00011 from mm import MarkovModel
00012 from importer import import_markopy
00013 markopy = import_markopy()
00014
00015 from base import BaseCLI,AbstractGenerationModelCLI, AbstractTrainingModelCLI
00016
00017 class MarkovPasswordsCLI(AbstractTrainingModelCLI,MarkovModel):
00018     """!
00019         @brief Extension of Python.Markopy.Base.BaseCLI for Markov::API::MarkovPasswords
00020         @belongsto Python::Markopy
00021         @extends Python::Markopy::MarkovModel
00022         @extends Python::Markopy::AbstractTrainingModelCLI
00023
00024         adds -st/--stdout arguement to the command line.
00025     """
00026     def __init__(self, add_help:bool=True):
00027         "! @brief initialize model with Markov::API::MarkovPasswords"
00028         super().__init__(add_help)
00029         self.modelmodelmodel = markopy.MarkovPasswords()
00030
00031     def _generate(self, wordlist):
00032         "! @brief map generation function to Markov::API::MarkovPasswords::Generate"
00033         self.modelmodelmodel.Generate(int(self.argsargs.count), wordlist, int(self.argsargs.min),
      int(self.argsargs.max), int(self.argsargs.threads))
00034
00035 if __name__ == "__main__":
00036     mp = MarkovPasswordsCLI()
00037     mp.parse()
00038     mp.process()
```

9.35 Markopy/Markopy/src/Module/markopy.cpp File Reference

CPython wrapper for libmarkov utils.

```
#include <Python.h>
#include <boost/python.hpp>
#include <MarkovAPI/src/modelMatrix.h>
```

Include dependency graph for markopy.cpp:



Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::Markopy

  CPython module for Markov::API objects.

Macros

- #define BOOST_ALL_STATIC_LIB 1
- #define BOOST_PYTHON_STATIC_LIB 1

Functions

- Markov::Markopy::BOOST_PYTHON_MODULE (markopy)

9.35.1 Detailed Description

CPython wrapper for libmarkov utils.

Authors

Ata Hakçıl, Celal Sahir Çetiner

This file is a wrapper for libmarkov utilities, exposing:

- MarkovPasswords
  - Import
  - Export
  - Train

  - Generate
- ModelMatrix
  - Import
  - Export
  - Train
  - ConstructMatrix
  - DumpJSON
  - FastRandomWalk

Markov::Model with char represented nodes. Includes wrappers for Markov::Model and additional helper functions to handle file I/O
This class is an extension of Markov::Model<char>, with higher level abstractions such as train and generate.
Class to flatten and reduce Markov::Model to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)
To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are sychronized and files are flushed every 50M operations.
Definition in file markopy.cpp.

9.35.2 Macro Definition Documentation

9.35.2.1 BOOST_ALL_STATIC_LIB

```
#define BOOST_ALL_STATIC_LIB 1
```
Definition at line 24 of file markopy.cpp.

9.35.2.2 BOOST_PYTHON_STATIC_LIB

```
#define BOOST_PYTHON_STATIC_LIB 1
```
Definition at line 25 of file markopy.cpp.

9.36 markopy.cpp

```
00001 /** @file markopy.cpp
00002  * @brief CPython wrapper for libmarkov utils.
00003  * @authors Ata Hakçıl, Celal Sahir Çetiner
00004  *
00005  * This file is a wrapper for libmarkov utilities, exposing:
00006  * - MarkovPasswords
00007  *   - Import
00008  *   - Export
00009  *   - Train
00010  *   - Generate
00011  * - ModelMatrix
00012  *   - Import
00013  *   - Export
00014  *   - Train
00015  *   - ConstructMatrix
00016  *   - DumpJSON
00017  *   - FastRandomWalk
00018  *
00019  * @copydoc Markov::API::MarkovPasswords
00020  * @copydoc Markov::API::ModelMatrix
00021  *
00022 */
00023
00024 #define BOOST_ALL_STATIC_LIB 1
00025 #define BOOST_PYTHON_STATIC_LIB 1
00026 #include <Python.h>
00027 #include <boost/python.hpp>
00028 #include <MarkovAPI/src/modelMatrix.h>
00029
```

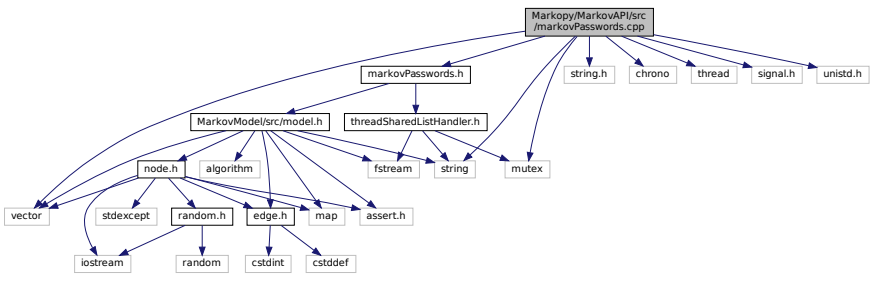
```
00030
00031 using namespace boost::python;
00032
00033 /**
00034  * @brief CPython module for Markov::API objects
00035  */
00036 namespace Markov::Markopy{
00037     BOOST_PYTHON_MODULE(markopy)
00038     {
00039         bool (Markov::API::MarkovPasswords::*Import)(const char*) = &Markov::Model<char>::Import;
00040         bool (Markov::API::MarkovPasswords::*Export)(const char*) = &Markov::Model<char>::Export;
00041         class_<Markov::API::MarkovPasswords>("MarkovPasswords", init<>())
00042             .def(init<>())
00043             .def("Train", &Markov::API::MarkovPasswords::Train)
00044             .def("Generate", &Markov::API::MarkovPasswords::Generate)
00045             .def("Import", Import, "Import a model file.")
00046             .def("Export", Export, "Export a model to file.")
00047         ;
00048
00049         int (Markov::API::ModelMatrix::*FastRandomWalk)(unsigned long int, const char*, int, int, int,
      bool)
00050             = &Markov::API::ModelMatrix::FastRandomWalk;
00051         class_<Markov::API::ModelMatrix>("ModelMatrix", init<>())
00052
00053             .def(init<>())
00054             .def("Train", &Markov::API::ModelMatrix::Train)
00055             .def("Import", &Markov::API::ModelMatrix::Import, "Import a model file.")
00056             .def("Export", Export, "Export a model to file.")
00057             .def("ConstructMatrix",&Markov::API::ModelMatrix::ConstructMatrix)
00058             .def("DumpJSON",&Markov::API::ModelMatrix::DumpJSON)
00059             .def("FastRandomWalk",FastRandomWalk)
00060             ;
00061     };
00062 };
```

9.37 Markopy/MarkovAPI/src/markovPasswords.cpp File Reference

Wrapper for Markov::Model to use with char represented models.

```
#include "markovPasswords.h"
#include <string.h>
#include <chrono>
#include <thread>
#include <vector>
#include <mutex>
#include <string>
#include <signal.h>
#include <unistd.h>
```

Include dependency graph for markovPasswords.cpp:



Functions

- void intHandler (int dummy)

Variables

- static volatile int keepRunning = 1

9.37.1 Detailed Description

Wrapper for Markov::Model to use with char represented models.

Authors

Ata Hakçıl, Osman Ömer Yıldıztugay

This file contains the implementation for Markov::API::MarkovPasswords class.
Markov::Model with char represented nodes. Includes wrappers for Markov::Model and additional helper functions to handle file I/O
This class is an extension of Markov::Model<char>, with higher level abstractions such as train and generate.
Definition in file markovPasswords.cpp.

9.37.2 Function Documentation

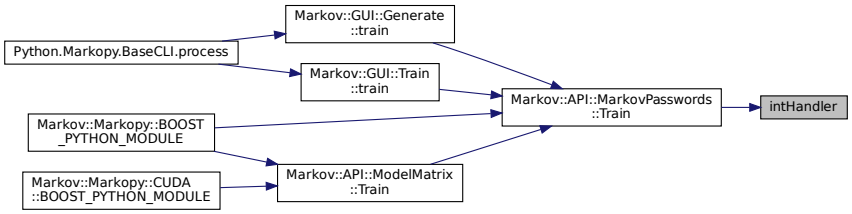
9.37.2.1 intHandler()

```
void intHandler (
            int dummy )
```

Definition at line 26 of file markovPasswords.cpp.

```
00026                           {
00027     std::cout « "Terminating.\n";
00028     //Sleep(5000);
00029     keepRunning = 0;
00030     exit(0);
00031 }
```

References keepRunning.
Referenced by Markov::API::MarkovPasswords::Train().
Here is the caller graph for this function:



9.37.3 Variable Documentation

9.37.3.1 keepRunning

```
volatile int keepRunning = 1  [static]
```

Definition at line 24 of file markovPasswords.cpp.
Referenced by intHandler(), and Markov::API::MarkovPasswords::TrainThread().

9.38 markovPasswords.cpp

```
00001 /** @file markovPasswords.cpp
00002  * @brief Wrapper for Markov::Model to use with char represented models.
00003  * @authors Ata Hakçıl, Osman Ömer Yıldıztugay
00004  *
00005  * This file contains the implementation for Markov::API::MarkovPasswords class.
00006  *
00007  * @copydoc Markov::API::MarkovPasswords
00008  */
```

```
00009
00010 #include "markovPasswords.h"
00011 #include <string.h>
00012 #include <chrono>
00013 #include <thread>
00014 #include <vector>
00015 #include <mutex>
00016 #include <string>
00017 #include <signal.h>
00018 #ifdef _WIN32
00019 #include <Windows.h>
00020 #else
00021 #include <unistd.h>
00022 #endif
00023
00024 static volatile int keepRunning = 1;
00025
00026 void intHandler(int dummy) {
00027     std::cout « "Terminating.\n";
00028     //Sleep(5000);
00029     keepRunning = 0;
00030     exit(0);
00031 }
00032
00033
00034 Markov::API::MarkovPasswords::MarkovPasswords() : Markov::Model<char>(){
00035
00036
00037 }
00038
00039 Markov::API::MarkovPasswords::MarkovPasswords(const char* filename) {
00040
00041     std::ifstream* importFile;
00042
00043     this->Import(filename);
00044
00045     //std::ifstream* newFile(filename);
00046
00047     //importFile = newFile;
00048
00049 }
00050
00051 std::ifstream* Markov::API::MarkovPasswords::OpenDatasetFile(const char* filename){
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
00062
00063
00064
00065 void Markov::API::MarkovPasswords::Train(const char* datasetFileName, char delimiter, int threads)   {
00066     signal(SIGINT, intHandler);
00067     Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
00068     auto start = std::chrono::high_resolution_clock::now();
00069
00070     std::vector<std::thread*> threadsV;
00071     for(int i=0;i<threads;i++){
00072         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
       &listhandler, delimiter));
00073     }
00074
00075     for(int i=0;i<threads;i++){
00076         threadsV[i]->join();
00077         delete threadsV[i];
00078     }
00079     auto finish = std::chrono::high_resolution_clock::now();
00080     std::chrono::duration<double> elapsed = finish - start;
00081     std::cout « "Elapsed time: " « elapsed.count() « " s\n";
00082
00083 }
00084
00085 void Markov::API::MarkovPasswords::TrainThread(Markov::API::Concurrency::ThreadSharedListHandler
       *listhandler, char delimiter){
00086     char format_str[] ="%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
```

```
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
       "%ld,%s"
00097 #else
00098         sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100         this->AdjustEdge((const char*)linebuf, oc);
00101         delete linebuf;
00102     }
00103 }
00104
00105
00106 std::ofstream* Markov::API::MarkovPasswords::Save(const char* filename) {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
00116
00117
00118 void Markov::API::MarkovPasswords::Generate(unsigned long int n, const char* wordlistFileName, int
       minLen, int maxLen, int threads)  {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++){
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
       &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00129     }
00130
00131     for(int i=0;i<threads;i++){
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
00139
00140 void Markov::API::MarkovPasswords::GenerateThread(std::mutex *outputLock, unsigned long int n,
       std::ofstream *wordlist, int minLen, int maxLen)  {
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist « res « "\n";
00149         outputLock->unlock();
00150     }
00151 }
00152
00153 void Markov::API::MarkovPasswords::Buff(const char* str, double multiplier, bool bDontAdjustSelfLoops,
       bool bDontAdjustExtendedLoops){
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173
00174         }
00175         i++;
```

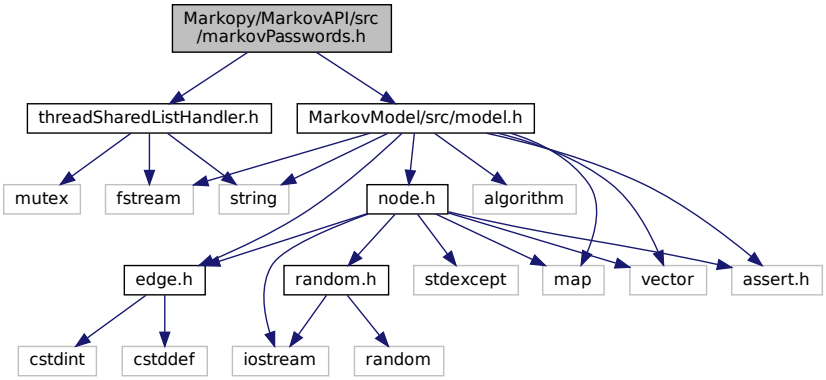
```
00176     }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

9.39 Markopy/MarkovAPI/src/markovPasswords.h File Reference

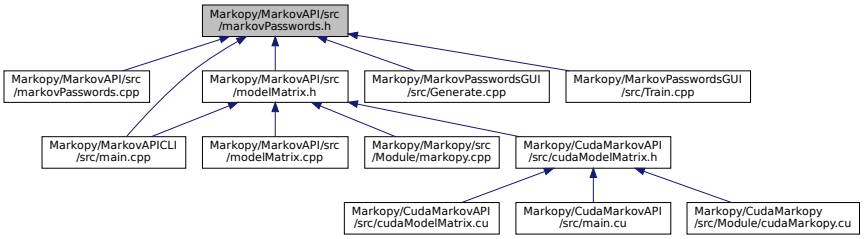
Wrapper for Markov::Model to use with char represented models.

```
#include "threadSharedListHandler.h"
#include "MarkovModel/src/model.h"
```

Include dependency graph for markovPasswords.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::API::MarkovPasswords

  Markov::Model with char represented nodes.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.

9.39.1 Detailed Description

Wrapper for Markov::Model to use with char represented models.

Authors

Ata Hakçıl, Osman Ömer Yıldıztugay

This file contains the declerations for Markov::API::MarkovPasswords class.
Markov::Model with char represented nodes. Includes wrappers for Markov::Model and additional helper functions to handle file I/O
This class is an extension of Markov::Model<char>, with higher level abstractions such as train and generate.
Definition in file markovPasswords.h.

9.40 markovPasswords.h

```
00001 /** @file markovPasswords.h
00002  * @brief Wrapper for Markov::Model to use with char represented models.
00003  * @authors Ata Hakçıl, Osman Ömer Yıldıztugay
00004  *
00005  * This file contains the declerations for Markov::API::MarkovPasswords class.
00006  *
00007  * @copydoc Markov::API::MarkovPasswords
00008  */
00009
00010 #pragma once
00011 #include "threadSharedListHandler.h"
00012 #include "MarkovModel/src/model.h"
00013
00014
00015 /** @brief Namespace for the MarkovPasswords API
00016 */
00017 namespace Markov::API{
00018
00019     /** @brief Markov::Model with char represented nodes.
00020      *
00021      * Includes wrappers for Markov::Model and additional helper functions to handle file I/O
00022      *
00023      * This class is an extension of Markov::Model<char>, with higher level abstractions such as train
       and generate.
00024      *
00025     */
00026     class MarkovPasswords : public Markov::Model<char>{
00027     public:
00028
00029         /** @brief Initialize the markov model from MarkovModel::Markov::Model.
00030          *
00031          * Parent constructor. Has no extra functionality.
00032         */
00033         MarkovPasswords();
00034
00035         /** @brief Initialize the markov model from MarkovModel::Markov::Model, with an import file.
00036          *
00037          * This function calls the Markov::Model::Import on the filename to construct the model.
00038          * Same thing as creating and empty model, and calling MarkovPasswords::Import on the
       filename.
00039          *
00040          * @param filename - Filename to import
00041          *
00042          *
00043          * @b Example @b Use: Construction via filename
00044          * @code{.cpp}
00045          * MarkovPasswords mp("test.mdl");
00046          * @endcode
00047         */
00048         MarkovPasswords(const char* filename);
00049
00050         /** @brief Open dataset file and return the ifstream pointer
00051          * @param filename - Filename to open
00052          * @return ifstream* to the the dataset file
00053         */
00054         std::ifstream* OpenDatasetFile(const char* filename);
00055
00056
00057         /** @brief Train the model with the dataset file.
00058          * @param datasetFileName - Ifstream* to the dataset. If null, use class member
00059          * @param delimiter - a character, same as the delimiter in dataset content
00060          * @param threads - number of OS threads to spawn
00061          *
00062          * @code{.cpp}
00063          * Markov::API::MarkovPasswords mp;
00064          * mp.Import("models/2gram.mdl");
00065          * mp.Train("password.corpus");
00066          * @endcode
00067         */
00068         void Train(const char* datasetFileName, char delimiter, int threads);
```

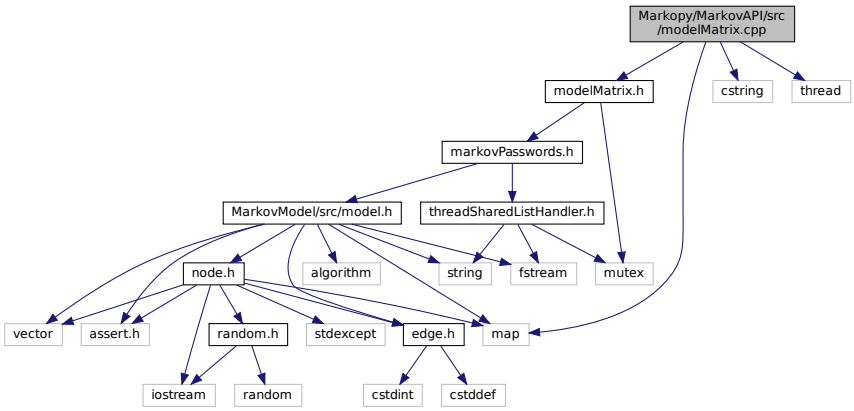
```
00069
00070
00071
00072         /** @brief Export model to file.
00073         * @param filename - Export filename.
00074         * @return std::ofstream* of the exported file.
00075         */
00076         std::ofstream* Save(const char* filename);
00077
00078         /** @brief Call Markov::Model::RandomWalk n times, and collect output.
00079          *
00080          * Generate from model and write results to a file.
00081          * a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5
      on average.
00082          *
00083          * @deprecated See Markov::API::MatrixModel::FastRandomWalk for more information.
00084          * @param n - Number of passwords to generate.
00085          * @param wordlistFileName - Filename to write to
00086          * @param minLen - Minimum password length to generate
00087          * @param maxLen - Maximum password length to generate
00088          * @param threads - number of OS threads to spawn
00089         */
00090         void Generate(unsigned long int n, const char* wordlistFileName, int minLen=6, int maxLen=12,
      int threads=20);
00091
00092         /** @brief Buff expression of some characters in the model
00093          * @param str A string containing all the characters to be buffed
00094          * @param multiplier A constant value to buff the nodes with.
00095          * @param bDontAdjustSelfEdges Do not adjust weights if target node is same as source node
00096          * @param bDontAdjustExtendedLoops Do not adjust if both source and target nodes are in first
      parameter
00097         */
00098         void Buff(const char* str, double multiplier, bool bDontAdjustSelfLoops=true, bool
      bDontAdjustExtendedLoops=false);
00099
00100
00101     private:
00102
00103         /** @brief A single thread invoked by the Train function.
00104          * @param listhandler - Listhandler class to read corpus from
00105          * @param delimiter - a character, same as the delimiter in dataset content
00106          *
00107         */
00108         void TrainThread(Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char
      delimiter);
00109
00110         /** @brief A single thread invoked by the Generate function.
00111          *
00112          * @b DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information.
      This has been replaced with
00113          * a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5
      on average.
00114          *
00115          * @param outputLock - shared mutex lock to lock during output operation. Prevents race
      condition on write.
00116          * @param n number of lines to be generated by this thread
00117          * @param wordlist wordlistfile
00118          * @param minLen - Minimum password length to generate
00119          * @param maxLen - Maximum password length to generate
00120          *
00121         */
00122         void GenerateThread(std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int
      minLen, int maxLen);
00123         std::ifstream* datasetFile; /** @brief  Dataset file input of our system    */
00124         std::ofstream* modelSavefile; /** @brief    File to save model  of our system   */
00125         std::ofstream* outputFile; /** @brief   Generated output  file  of our system   */
00126     };
00127
00128
00129
00130 };
```

9.41 Markopy/MarkovAPI/src/modelMatrix.cpp File Reference

An extension of Markov::API::MarkovPasswords.

```
#include "modelMatrix.h"
#include <map>
#include <cstring>
#include <thread>
```

Include dependency graph for modelMatrix.cpp:



9.41.1 Detailed Description

An extension of Markov::API::MarkovPasswords.

Authors

Ata Hakçıl

This class shows superior performance compared to the traditional model at Markov::API::MarkovPasswords
Class to flatten and reduce Markov::Model to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)
To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are sychronized and files are flushed every 50M operations.
Definition in file modelMatrix.cpp.

9.42 modelMatrix.cpp

```
00001 /** @file modelMatrix.cpp
00002  * @brief An extension of Markov::API::MarkovPasswords
00003  * @authors Ata Hakçıl
00004  *
00005  * This class shows superior performance compared to the traditional model at
       Markov::API::MarkovPasswords
00006  *
00007  * @copydoc Markov::API::ModelMatrix
00008  */
00009
00010 #include "modelMatrix.h"
00011 #include <map>
00012 #include <cstring>
00013 #include <thread>
00014
00015 Markov::API::ModelMatrix::ModelMatrix(){
00016     this->ready = false;
00017 }
00018
00019 void Markov::API::ModelMatrix::Import(const char *filename){
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
00024
00025 void Markov::API::ModelMatrix::Train(const char *datasetFileName, char delimiter, int threads){
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
00030
00031 bool Markov::API::ModelMatrix::ConstructMatrix(){
00032     if(this->ready) return false;
```

```
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j]  = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072
00073         }
00074         i++;
00075     }
00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
00080
00081 bool Markov::API::ModelMatrix::DeallocateMatrix(){
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
00085
00086     for(int i=0;i<this->matrixSize;i++){
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090
00091     for(int i=0;i<this->matrixSize;i++){
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
00099 }
00100
00101 void Markov::API::ModelMatrix::DumpJSON(){
00102
00103     std::cout << "{\n   \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout << "\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout << "\\\\";
00107         else if(this->matrixIndex[i]==0) std::cout << "\\\\x00";
00108         else if(i==0) std::cout << "\\\\xff";
00109         else if(this->matrixIndex[i]=='\n') std::cout << "\\n";
00110         else std::cout << this->matrixIndex[i];
00111     }
00112     std::cout <<
00113     "\",\n"
00114     "   \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++){
00117         if(this->matrixIndex[i]=='"') std::cout << "      \"\\\"\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\x00\": [";
```

```
00120         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00121         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00122         for(int j=0;j<this->matrixSize;j++){
00123             if(this->edgeMatrix[i][j]=='"') std::cout « "\"\\\"\"";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout « "\"\\\\\"";
00125             else if(this->edgeMatrix[i][j]==0) std::cout « "\"\\\\x00\"";
00126             else if(this->edgeMatrix[i][j]<0) std::cout « "\"\\\\xff\"";
00127             else if(this->matrixIndex[i]=='\n') std::cout « "\"\\n\"";
00128             else std::cout « "\"" « this->edgeMatrix[i][j] « "\"";
00129             if(j!=this->matrixSize-1) std::cout « ", ";
00130         }
00131         std::cout « "],\n";
00132     }
00133     std::cout « "},\n";
00134
00135     std::cout « "\"   weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++){
00137         if(this->matrixIndex[i]=='"') std::cout « "      \"\\\"\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout « "      \"\\\\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout « "      \"\\\\x00\": [";
00140         else if(this->matrixIndex[i]<0) std::cout « "      \"\\\\xff\": [";
00141         else std::cout « "      \"" « this->matrixIndex[i] « "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++){
00144             std::cout « this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout « ", ";
00146         }
00147         std::cout « "],\n";
00148     }
00149     std::cout « "  }\n}\n";
00150 }
00151
00152
00153 void Markov::API::ModelMatrix::FastRandomWalkThread(std::mutex *mlock, std::ofstream *wordlist,
      unsigned long int n, int minLen, int maxLen, int id, bool bFileIO){
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179
00180             if (len >= maxLen)  break;
00181             else if ((next < 0) && (len < minLen)) continue;
00182             else if (next < 0) break;
00183             cur = next;
00184             res[bufferctr + len++] = cur;
00185         }
00186         res[bufferctr + len++] = '\n';
00187         bufferctr+=len;
00188
00189     }
00190     if(bFileIO){
00191         mlock->lock();
00192         *wordlist « res;
00193         mlock->unlock();
00194     }else{
00195         mlock->lock();
00196         std::cout « res;
00197         mlock->unlock();
00198     }
00199     delete res;
00200
00201 }
00202
00203
00204 int Markov::API::ModelMatrix::FastRandomWalk(unsigned long int n, std::ofstream *wordlist, int minLen,
      int maxLen, int threads, bool bFileIO){
```

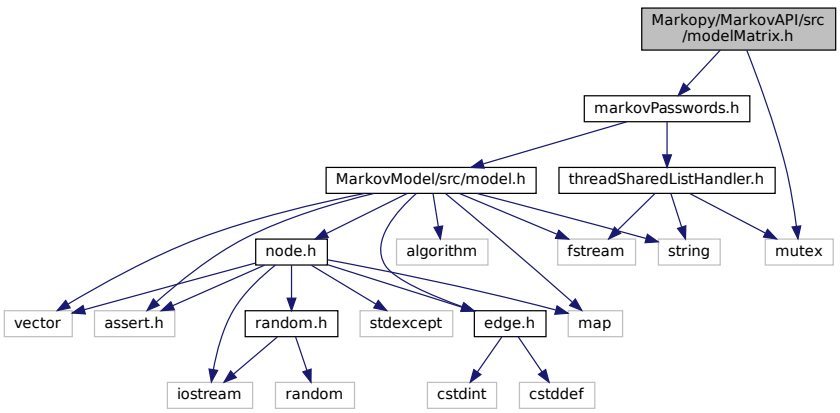
```
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
      threads);
00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
      threads);
00213     }
00214     return 0;
00215 }
00216
00217 int Markov::API::ModelMatrix::FastRandomWalk(unsigned long int n, const char* wordlistFileName, int
      minLen, int maxLen, int threads, bool bFileIO){
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
00224
00225 void Markov::API::ModelMatrix::FastRandomWalkPartition(std::mutex *mlock, std::ofstream *wordlist,
      unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads){
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++){
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
      mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
      wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00239
00240     for(int i=0;i<threads;i++){
00241         threadsV[i]->join();
00242     }
00243 }
```

9.43 Markopy/MarkovAPI/src/modelMatrix.h File Reference

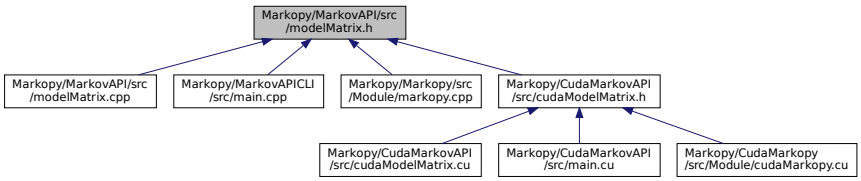
An extension of Markov::API::MarkovPasswords.

```
#include "markovPasswords.h"
#include <mutex>
```

Include dependency graph for modelMatrix.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::API::ModelMatrix

  Class to flatten and reduce Markov::Model to a Matrix.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.

9.43.1 Detailed Description

An extension of Markov::API::MarkovPasswords.

Authors

Ata Hakçıl

This class shows superior performance compared to the traditional model at Markov::API::MarkovPasswords
Class to flatten and reduce Markov::Model to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)
To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are sychronized and files are flushed every 50M operations.
Definition in file modelMatrix.h.

9.44 modelMatrix.h

```
00001 /** @file modelMatrix.h
00002  * @brief An extension of Markov::API::MarkovPasswords
00003  * @authors Ata Hakçıl
00004  *
00005  * This class shows superior performance compared to the traditional model at
      Markov::API::MarkovPasswords
00006  *
00007  * @copydoc Markov::API::ModelMatrix
00008  *
00009  */
00010
00011 #include "markovPasswords.h"
00012 #include <mutex>
00013
00014 namespace Markov::API{
00015
00016     /** @brief Class to flatten and reduce Markov::Model to a Matrix
00017      *
00018      * Matrix level operations can be used for Generation events, with a significant performance
      optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)
00019      *
00020      * To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for
      allocation. Threads are sychronized and files are flushed every 50M operations.
00021      *
```

```
00022     */
00023     class ModelMatrix : public Markov::API::MarkovPasswords{
00024     public:
00025         ModelMatrix();
00026 
00027         /** @brief Construct the related Matrix data for the model.
00028          *
00029          * This operation can be used after importing/training to allocate and populate the matrix
      content.
00030          *
00031          * this will initialize:
00032          * char** edgeMatrix -> a 2D array of mapping left and right connections of each edge.
00033          * long int **valueMatrix -> a 2D array representing the edge weights.
00034          * int matrixSize -> Size of the matrix, aka total number of nodes.
00035          * char* matrixIndex -> order of nodes in the model
00036          * long int *totalEdgeWeights -> total edge weights of each Node.
00037          *
00038          * @returns True if constructed. False if already construced.
00039          */
00040         bool ConstructMatrix();
00041 
00042 
00043         /** @brief Debug function to dump the model to a JSON file.
00044          *
00045          * Might not work 100%. Not meant for production use.
00046          */
00047         void DumpJSON();
00048 
00049 
00050         /** @brief Random walk on the Matrix-reduced Markov::Model
00051          *
00052          * This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are
      partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
00053          *
00054          * If n>50M, threads are going to be synced, files are going to be flushed, and buffers will
      be reallocated every 50M generations.
00055          * This comes at a minor performance penalty.
00056          *
00057          * While it has the same functionality, this operation reduces
      Markov::API::MarkovPasswords::Generate runtime by %96.5
00058          *
00059          * This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually
      replace it.
00060          *
00061          * @param n - Number of passwords to generate.
00062          * @param wordlistFileName - Filename to write to
00063          * @param minLen - Minimum password length to generate
00064          * @param maxLen - Maximum password length to generate
00065          * @param threads - number of OS threads to spawn
00066          * @param bFileIO - If false, filename will be ignored and will output to stdout.
00067          *
00068          *
00069          * @code{.cpp}
00070          * Markov::API::ModelMatrix mp;
00071          * mp.Import("models/finished.mdl");
00072          * mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
00073          * @endcode
00074          *
00075          */
00076         int FastRandomWalk(unsigned long int n, const char* wordlistFileName, int minLen=6, int
      maxLen=12, int threads=20, bool bFileIO=true);
00077 
00078         /** @copydoc Markov::Model::Import(const char *filename)
00079          * Construct the matrix when done.
00080          *
00081          */
00082         void Import(const char *filename);
00083 
00084         /** @copydoc Markov::API::MarkovPasswords::Train(const char *datasetFileName, char delimiter,
      int threads)
00085          * Construct the matrix when done.
00086          *
00087          */
00088         void Train(const char *datasetFileName, char delimiter, int threads);
00089 
00090     protected:
00091 
00092         /** @brief Random walk on the Matrix-reduced Markov::Model
00093          *
00094          * This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are
      partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
00095          *
00096          * If n>50M, threads are going to be synced, files are going to be flushed, and buffers will
      be reallocated every 50M generations.
00097          * This comes at a minor performance penalty.
00098          *
00099          * While it has the same functionality, this operation reduces
```

```
      Markov::API::MarkovPasswords::Generate runtime by %96.5
00100          *
00101          * This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually
      replace it.
00102          *
00103          * @param n - Number of passwords to generate.
00104          * @param wordlistFileName - Filename to write to
00105          * @param minLen - Minimum password length to generate
00106          * @param maxLen - Maximum password length to generate
00107          * @param threads - number of OS threads to spawn
00108          * @param bFileIO - If false, filename will be ignored and will output to stdout.
00109          *
00110          *
00111          * @code{.cpp}
00112          * Markov::API::ModelMatrix mp;
00113          * mp.Import("models/finished.mdl");
00114          * mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
00115          * @endcode
00116          *
00117         */
00118         int FastRandomWalk(unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12,
      int threads=20, bool bFileIO=true);
00119
00120
00121         /** @brief A single partition of FastRandomWalk event
00122          *
00123          * Since FastRandomWalk has to allocate its output buffer before operation starts and writes
      data in chunks,
00124          * large n parameters would lead to huge memory allocations.
00125          * @b Without @b Partitioning:
00126          * - 50M results 12 characters max -> 550 Mb Memory allocation
00127          *
00128          * - 5B results  12 characters max -> 55 Gb Memory allocation
00129          *
00130          * - 50B results 12 characters max -> 550GB Memory allocation
00131          *
00132          * Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.
00133          *
00134          * @param mlock - mutex lock to distribute to child threads
00135          * @param wordlist - Reference to the wordlist file to write to
00136          * @param n - Number of passwords to generate.
00137          * @param wordlistFileName - Filename to write to
00138          * @param minLen - Minimum password length to generate
00139          * @param maxLen - Maximum password length to generate
00140          * @param threads - number of OS threads to spawn
00141          * @param bFileIO - If false, filename will be ignored and will output to stdout.
00142          *
00143          *
00144         */
00145         void FastRandomWalkPartition(std::mutex *mlock, std::ofstream *wordlist, unsigned long int n,
      int minLen, int maxLen, bool bFileIO, int threads);
00146
00147         /** @brief A single thread of a single partition of FastRandomWalk
00148          *
00149          * A FastRandomWalkPartition will initiate as many of this function as requested.
00150          *
00151          * This function contains the bulk of the generation algorithm.
00152          *
00153          * @param mlock - mutex lock to distribute to child threads
00154          * @param wordlist - Reference to the wordlist file to write to
00155          * @param n - Number of passwords to generate.
00156          * @param wordlistFileName - Filename to write to
00157          * @param minLen - Minimum password length to generate
00158          * @param maxLen - Maximum password length to generate
00159          * @param id - @b DEPRECATED Thread id - No longer used
00160          * @param bFileIO - If false, filename will be ignored and will output to stdout.
00161          *
00162          *
00163         */
00164         void FastRandomWalkThread(std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int
      minLen, int maxLen, int id, bool bFileIO);
00165
00166         /** @brief Deallocate matrix and make it ready for re-construction
00167          *
00168          * @returns True if deallocated. False if matrix was not initialized
00169         */
00170         bool DeallocateMatrix();
00171
00172         /**
00173             @brief 2-D Character array for the edge Matrix (The characters of Nodes)
00174         */
00175         char** edgeMatrix;
00176
00177         /**
00178             @brief 2-d Integer array for the value Matrix (For the weights of  Edges)
00179         */
00180         long int **valueMatrix;
```

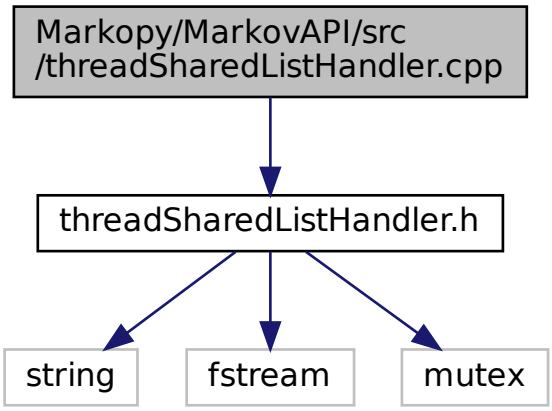
```
00181
00182         /**
00183             @brief to hold Matrix size
00184         */
00185         int matrixSize;
00186
00187         /**
00188             @brief to hold the Matrix index (To hold the orders of 2-D arrays')
00189         */
00190         char* matrixIndex;
00191
00192         /**
00193             @brief Array of the Total Edge Weights
00194         */
00195         long int *totalEdgeWeights;
00196
00197         /**
00198             @brief True when matrix is constructed. False if not.
00199         */
00200         bool ready;
00201     };
00202
00203
00204
00205 };
```

9.45 Markopy/MarkovAPI/src/threadSharedListHandler.cpp File Reference

Thread-safe wrapper for std::ifstream.

`#include "threadSharedListHandler.h"`

Include dependency graph for threadSharedListHandler.cpp:



9.45.1 Detailed Description

Thread-safe wrapper for std::ifstream.

Authors

Ata Hakçıl

Simple class for managing shared access to file. This class maintains the handover of each line from a file to multiple threads.
When two different threads try to read from the same file while reading a line isn't completed, it can have unexpected results. Line might be split, or might be read twice. This class locks the read action on the list until a line is completed, and then proceeds with the handover.

Definition in file threadSharedListHandler.cpp.

9.46 threadSharedListHandler.cpp

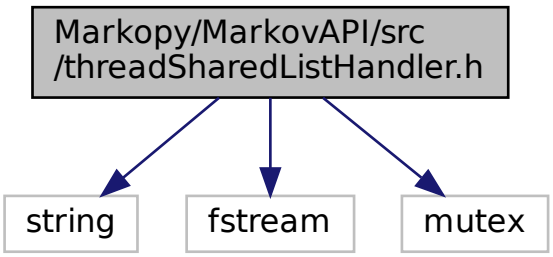
```
00001 /** @file threadSharedListHandler.cpp
00002  * @brief Thread-safe wrapper for std::ifstream
00003  * @authors Ata Hakçıl
00004  *
00005  * @copydoc Markov::API::Concurrency::ThreadSharedListHandler
00006  *
00007  */
00008
00009 #include "threadSharedListHandler.h"
00010
00011
00012 Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler(const char* filename){
00013     this->listfile;
00014     this->listfile.open(filename, std::ios_base::binary);
00015 }
00016
00017
00018 bool Markov::API::Concurrency::ThreadSharedListHandler::next(std::string* line){
00019     bool res = false;
00020     this->mlock.lock();
00021     res = (std::getline(this->listfile,*line,'\n'))? true : false;
00022     this->mlock.unlock();
00023
00024     return res;
00025 }
```

9.47 Markopy/MarkovAPI/src/threadSharedListHandler.h File Reference

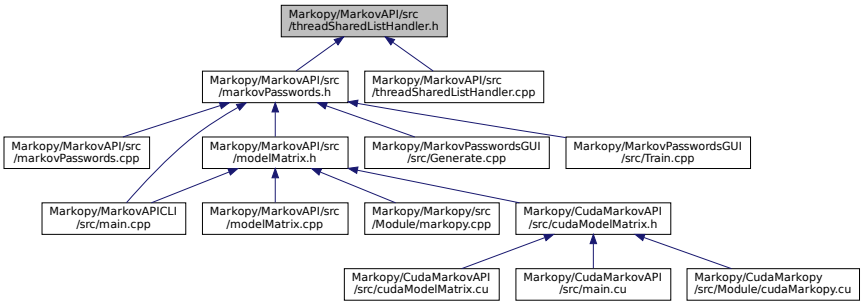
Thread-safe wrapper for std::ifstream.

```
#include <string>
#include <fstream>
#include <mutex>
```

Include dependency graph for threadSharedListHandler.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::API::Concurrency::ThreadSharedListHandler

  Simple class for managing shared access to file.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.
- Markov::API::Concurrency

  Namespace for Concurrency related classes.

9.47.1 Detailed Description

Thread-safe wrapper for std::ifstream.

Authors

Ata Hakçıl

Simple class for managing shared access to file. This class maintains the handover of each line from a file to multiple threads.
When two different threads try to read from the same file while reading a line isn't completed, it can have unexpected results. Line might be split, or might be read twice. This class locks the read action on the list until a line is completed, and then proceeds with the handover.
Definition in file threadSharedListHandler.h.

9.48 threadSharedListHandler.h

```
00001 /** @file threadSharedListHandler.h
00002  * @brief Thread-safe wrapper for std::ifstream
00003  * @authors Ata Hakçıl
00004  *
00005  * @copydoc Markov::API::Concurrency::ThreadSharedListHandler
00006  */
00007
00008 #include <string>
00009 #include <fstream>
00010 #include <mutex>
00011
00012 /** @brief Namespace for Concurrency related classes
00013 */
00014 namespace Markov::API::Concurrency{
00015
00016 /** @brief Simple class for managing shared access to file
00017  *
```

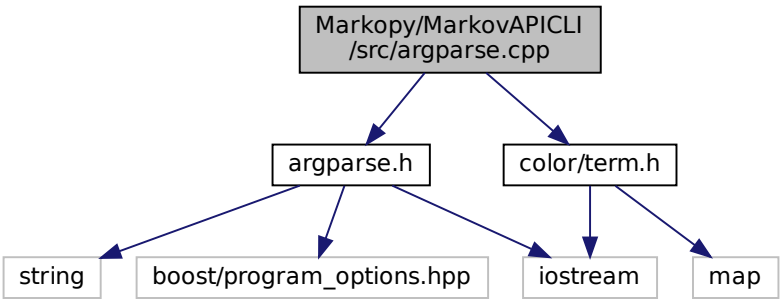
```
00018  * This class maintains the handover of each line from a file to multiple threads.
00019  *
00020  * When two different threads try to read from the same file while reading a line isn't completed, it
       can have unexpected results.
00021  * Line might be split, or might be read twice.
00022  * This class locks the read action on the list until a line is completed, and then proceeds with the
       handover.
00023  *
00024 */
00025 class ThreadSharedListHandler{
00026 public:
00027     /** @brief Construct the Thread Handler with a filename
00028      *
00029      * Simply open the file, and initialize the locks.
00030      *
00031      * @b Example @b Use: Simple file read
00032      * @code{.cpp}
00033      * ThreadSharedListHandler listhandler("test.txt");
00034      * std::string line;
00035      * std::cout « listhandler->next(&line) « "\n";
00036      * @endcode
00037      *
00038      * @b Example @b Use: Example use case from MarkovPasswords showing multithreaded access
00039      * @code{.cpp}
00040      *  void MarkovPasswords::Train(const char* datasetFileName, char delimiter, int threads)   {
00041      *       ThreadSharedListHandler listhandler(datasetFileName);
00042      *       auto start = std::chrono::high_resolution_clock::now();
00043      *
00044      *       std::vector<std::thread*> threadsV;
00045      *       for(int i=0;i<threads;i++){
00046      *           threadsV.push_back(new std::thread(&MarkovPasswords::TrainThread, this, &listhandler,
       datasetFileName, delimiter));
00047      *       }
00048      *
00049      *      for(int i=0;i<threads;i++){
00050      *           threadsV[i]->join();
00051      *           delete threadsV[i];
00052      *       }
00053      *       auto finish = std::chrono::high_resolution_clock::now();
00054      *       std::chrono::duration<double> elapsed = finish - start;
00055      *       std::cout « "Elapsed time: " « elapsed.count() « " s\n";
00056      *
00057      *   }
00058      *
00059      *   void MarkovPasswords::TrainThread(ThreadSharedListHandler *listhandler, const char*
       datasetFileName, char delimiter){
00060      *       char format_str[] ="%ld,%s";
00061      *       format_str[2]=delimiter;
00062      *       std::string line;
00063      *       while (listhandler->next(&line)) {
00064      *           long int oc;
00065      *           if (line.size() > 100) {
00066      *               line = line.substr(0, 100);
00067      *           }
00068      *           char* linebuf = new char[line.length()+5];
00069      *           sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
00070      *           this->AdjustEdge((const char*)linebuf, oc);
00071      *           delete linebuf;
00072      *       }
00073      *   }
00074      * @endcode
00075      *
00076      * @param filename Filename for the file to manage.
00077     */
00078     ThreadSharedListHandler(const char* filename);
00079
00080     /** @brief Read the next line from the file.
00081      *
00082      * This action will be blocked until another thread (if any) completes the read operation on the
       file.
00083      *
00084      * @b Example @b Use: Simple file read
00085      * @code{.cpp}
00086      * ThreadSharedListHandler listhandler("test.txt");
00087      * std::string line;
00088      * std::cout « listhandler->next(&line) « "\n";
00089      * @endcode
00090      *
00091     */
00092     bool next(std::string* line);
00093
00094 private:
00095     std::ifstream listfile;
00096     std::mutex mlock;
00097 };
00098
00099 };
```

9.49 Markopy/MarkovAPICLI/src/argparse.cpp File Reference

Arguement handler class for native CPP cli.

```
#include "argparse.h"
#include "color/term.h"
```

Include dependency graph for argparse.cpp:



9.49.1 Detailed Description

Arguement handler class for native CPP cli.

Authors

Celal Sahir Çetiner

Parse command line arguements.
Definition in file argparse.cpp.

9.50 argparse.cpp

```
00001 /** @file argparse.cpp
00002  * @brief Arguement handler class for native CPP cli
00003  * @authors Celal Sahir Çetiner
00004  *
00005  * @copydoc Markov::API::CLI::Argparse
00006  */
00007
00008 #include "argparse.h"
00009 #include "color/term.h"
00010
00011 Markov::API::CLI::ProgramOptions* Markov::API::CLI::Argparse::parse(int argc, char** argv) { return 0;
       }
00012
00013
00014
00015 void Markov::API::CLI::Argparse::help() {
00016     std::cout «
00017     "Markov Passwords - Help\n"
00018     "Options:\n"
00019     "    \n"
00020     "    -of --outputfilename\n"
00021     "        Filename to output the generation results\n"
00022     "    -ef --exportfilename\n"
00023     "        filename to export built model to\n"
00024     "    -if --importfilename\n"
00025     "        filename to import model from\n"
00026     "    -n (generate count)\n"
00027     "        Number of lines to generate\n"
00028     "    \n"
00029     "Usage: \n"
00030     "    markov.exe -if empty_model.mdl -ef model.mdl\n"
00031     "        import empty_model.mdl and train it with data from stdin. When done, output the model to
      model.mdl\n"
00032     "\n"
```

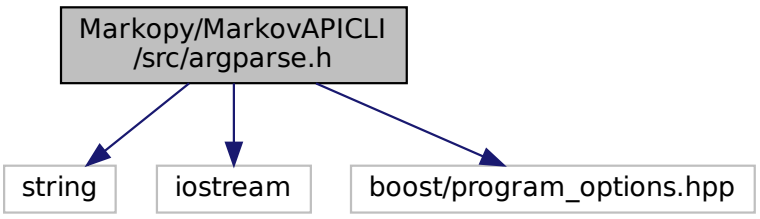
```
00033     "   markov.exe -if empty_model.mdl -n 15000 -of wordlist.txt\n"
00034     "       import empty_model.mdl and generate 15000 words to wordlist.txt\n"
00035
00036      « std::endl;
00037 }
```

9.51 Markopy/MarkovAPICLI/src/argparse.h File Reference

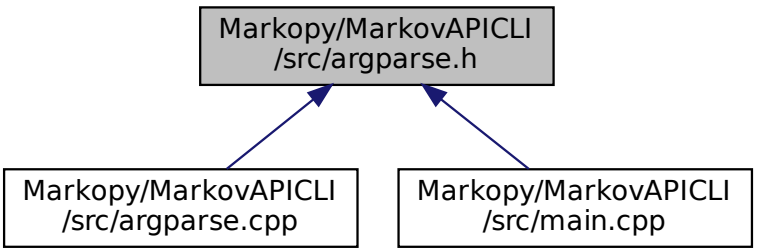
Arguement handler class for native CPP cli.

```
#include <string>
#include <iostream>
#include <boost/program_options.hpp>
```

Include dependency graph for argparse.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct Markov::API::CLI::_programOptions

  Structure to hold parsed cli 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_STATIC_LIB 1
- #define BOOST_PROGRAM_OPTIONS_STATIC_LIB 1

Typedefs

- typedef struct Markov::API::CLI::_programOptions Markov::API::CLI::ProgramOptions

  Structure to hold parsed cli arguements.

9.51.1 Detailed Description

Arguement handler class for native CPP cli.

Authors

Celal Sahir Çetiner

Definition in file argparse.h.

9.51.2 Macro Definition Documentation

9.51.2.1 BOOST_ALL_STATIC_LIB

```
#define BOOST_ALL_STATIC_LIB 1
```

Definition at line 11 of file argparse.h.

9.51.2.2 BOOST_PROGRAM_OPTIONS_STATIC_LIB

```
#define BOOST_PROGRAM_OPTIONS_STATIC_LIB 1
```

Definition at line 12 of file argparse.h.

9.52 argparse.h

```
00001 /** @file argparse.h
00002  * @brief Arguement handler class for native CPP cli
00003  * @authors Celal Sahir Çetiner
00004  *
00005  * @copydoc Markov::API::CLI::Argparse:
00006  */
00007
00008 #include<string>
00009 #include<iostream>
00010
00011 #define BOOST_ALL_STATIC_LIB 1
00012 #define BOOST_PROGRAM_OPTIONS_STATIC_LIB 1
00013
00014 #include <boost/program_options.hpp>
00015
00016 /** @brief Structure to hold parsed cli arguements.
00017 */
00018 namespace opt = boost::program_options;
00019
00020 /**
00021     @brief Namespace for the CLI objects
00022 */
00023 namespace Markov::API::CLI{
00024
00025     /** @brief Structure to hold parsed cli arguements. */
00026     typedef struct _programOptions {
```

```
00027         /**
00028             @brief Import flag to validate import
00029         */
00030         bool bImport;
00031
00032         /**
00033             @brief Export flag to validate export
00034         */
00035         bool bExport;
00036
00037         /**
00038             @brief Failure flag to validate succesfull running
00039         */
00040         bool bFailure;
00041
00042         /**
00043             @brief Seperator character to use with training data. (character between occurence and
      value)"
00044         */
00045         char seperator;
00046
00047         /**
00048             @brief Import name of our model
00049         */
00050         std::string importname;
00051
00052         /**
00053             @brief Import name of our given wordlist
00054         */
00055         std::string exportname;
00056
00057         /**
00058             @brief Import name of our given wordlist
00059         */
00060         std::string wordlistname;
00061
00062         /**
00063             @brief Output name of our generated password list
00064         */
00065         std::string outputfilename;
00066
00067         /**
00068             @brief The name of the given dataset
00069         */
00070         std::string datasetname;
00071
00072         /**
00073             @brief Number of passwords to be generated
00074         */
00075         int generateN;
00076
00077     } ProgramOptions;
00078
00079
00080     /** @brief Parse command line arguements
00081     */
00082     class Argparse {
00083     public:
00084
00085         Argparse();
00086
00087         /** @brief Parse command line arguements.
00088          *
00089          * Parses command line arguements to populate ProgramOptions structure.
00090          *
00091          * @param argc Number of command line arguements
00092          * @param argv Array of command line parameters
00093         */
00094         Argparse(int argc, char** argv) {
00095
00096             /*bool bImp;
00097             bool bExp;
00098             bool bFail;
00099             char sprt;
00100             std::string imports;
00101             std::string exports;
00102             std::string outputs;
00103             std::string datasets;
00104             int generateN;
00105             */
00106             opt::options_description desc("Options");
00107
00108
00109             desc.add_options()
00110                 ("generate", "Generate strings with given parameters")
00111                 ("train", "Train model with given parameters")
00112                 ("combine", "Combine")
```

```
00113                 ("import", opt::value<std::string>(), "Import model file")
00114                 ("output", opt::value<std::string>(), "Output model file. This model will be exported
    when done. Will be ignored for generation mode")
00115                 ("dataset", opt::value<std::string>(), "Dataset file to read input from training. Will
    be ignored for generation mode")
00116                 ("seperator", opt::value<char>(), "Seperator character to use with training data.
    (character between occurence and value)")
00117                 ("wordlist", opt::value<std::string>(), "Wordlist file path to export generation
    results to. Will be ignored for training mode")
00118                 ("count", opt::value<int>(), "Number of lines to generate. Ignored in training mode")
00119                 ("verbosity", "Output verbosity")
00120                 ("help", "Option definitions");
00121
00122                 opt::variables_map vm;
00123
00124                 opt::store(opt::parse_command_line(argc, argv, desc), vm);
00125
00126                 opt::notify(vm);
00127
00128                 //std::cout << desc << std::endl;
00129                 if (vm.count("help")) {
00130                 std::cout << desc << std::endl;
00131                 }
00132
00133                 if (vm.count("output") == 0) this->po.outputfilename = "NULL";
00134                 else if (vm.count("output") == 1) {
00135                     this->po.outputfilename = vm["output"].as<std::string>();
00136                     this->po.bExport = true;
00137                 }
00138                 else {
00139                     this->po.bFailure = true;
00140                     std::cout << "UNIDENTIFIED INPUT" << std::endl;
00141                     std::cout << desc << std::endl;
00142                 }
00143
00144
00145                 if (vm.count("dataset") == 0) this->po.datasetname = "NULL";
00146                 else if (vm.count("dataset") == 1) {
00147                     this->po.datasetname = vm["dataset"].as<std::string>();
00148                 }
00149                 else {
00150                     this->po.bFailure = true;
00151                     std::cout << "UNIDENTIFIED INPUT" << std::endl;
00152                     std::cout << desc << std::endl;
00153                 }
00154
00155
00156                 if (vm.count("wordlist") == 0) this->po.wordlistname = "NULL";
00157                 else if (vm.count("wordlist") == 1) {
00158                     this->po.wordlistname = vm["wordlist"].as<std::string>();
00159                 }
00160                 else {
00161                     this->po.bFailure = true;
00162                     std::cout << "UNIDENTIFIED INPUT" << std::endl;
00163                     std::cout << desc << std::endl;
00164                 }
00165
00166                 if (vm.count("import") == 0) this->po.importname = "NULL";
00167                 else if (vm.count("import") == 1) {
00168                     this->po.importname = vm["import"].as<std::string>();
00169                     this->po.bImport = true;
00170                 }
00171                 else {
00172                     this->po.bFailure = true;
00173                     std::cout << "UNIDENTIFIED INPUT" << std::endl;
00174                     std::cout << desc << std::endl;
00175                 }
00176
00177
00178                 if (vm.count("count") == 0) this->po.generateN = 0;
00179                 else if (vm.count("count") == 1) {
00180                     this->po.generateN = vm["count"].as<int>();
00181                 }
00182                 else {
00183                     this->po.bFailure = true;
00184                     std::cout << "UNIDENTIFIED INPUT" << std::endl;
00185                     std::cout << desc << std::endl;
00186                 }
00187
00188                 /*std::cout << vm["output"].as<std::string>() << std::endl;
00189                 std::cout << vm["dataset"].as<std::string>() << std::endl;
00190                 std::cout << vm["wordlist"].as<std::string>() << std::endl;
00191                 std::cout << vm["output"].as<std::string>() << std::endl;
00192                 std::cout << vm["count"].as<int>() << std::endl;*/
00193
00194
00195                 //else if (vm.count("train")) std::cout << "train oldu" << std::endl;
```

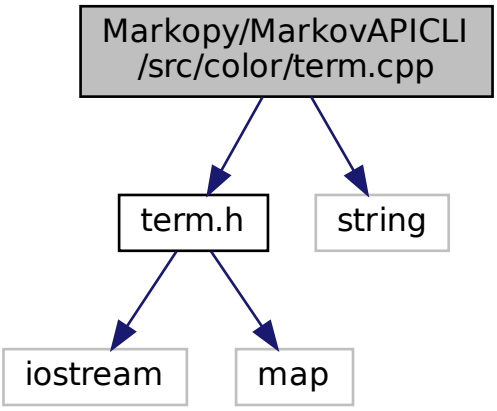
```
00196         }
00197
00198         /** @brief Getter for command line options
00199          *
00200          * Getter for ProgramOptions populated by the arguement parser
00201          * @returns ProgramOptions structure.
00202         */
00203         Markov::API::CLI::ProgramOptions getProgramOptions(void) {
00204             return this->po;
00205         }
00206
00207         /** @brief Initialize program options structure.
00208          *
00209          * @param i boolean, true if import operation is flagged
00210          * @param e boolean, true if export operation is flagged
00211          * @param bf boolean, true if there is something wrong with the command line parameters
00212          * @param s seperator character for the import function
00213          * @param iName import filename
00214          * @param exName export filename
00215          * @param oName output filename
00216          * @param dName corpus filename
00217          * @param n number of passwords to be generated
00218          *
00219         */
00220         void setProgramOptions(bool i, bool e, bool bf, char s, std::string iName, std::string exName,
      std::string oName, std::string dName, int n) {
00221             this->po.bImport = i;
00222             this->po.bExport = e;
00223             this->po.seperator = s;
00224             this->po.bFailure = bf;
00225             this->po.generateN = n;
00226             this->po.importname = iName;
00227             this->po.exportname = exName;
00228             this->po.outputfilename = oName;
00229             this->po.datasetname = dName;
00230
00231             /*strcpy_s(this->po.importname,256,iName);
00232             strcpy_s(this->po.exportname,256,exName);
00233             strcpy_s(this->po.outputfilename,256,oName);
00234             strcpy_s(this->po.datasetname,256,dName);*/
00235
00236         }
00237
00238         /** @brief parse cli commands and return
00239         * @param argc - Program arguement count
00240         * @param argv - Program arguement values array
00241         * @return ProgramOptions structure.
00242         */
00243         static Markov::API::CLI::ProgramOptions* parse(int argc, char** argv);
00244
00245
00246         /** @brief Print help string.
00247         */
00248         static void help();
00249
00250     private:
00251         /**
00252             @brief ProgramOptions structure object
00253         */
00254
00255         Markov::API::CLI::ProgramOptions po;
00256     };
00257
00258 };
```

9.53 Markopy/MarkovAPICLI/src/color/term.cpp File Reference

Terminal handler for pretty stuff like colors.

```
#include "term.h"
#include <string>
```

Include dependency graph for term.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const Terminal::color &c)

9.53.1 Detailed Description

Terminal handler for pretty stuff like colors.

Authors

Ata Hakçıl

pretty colors for Terminal. Windows Only.
Definition in file term.cpp.

9.53.2 Function Documentation

9.53.2.1 operator<<()

```
std::ostream& operator<< (
            std::ostream & os,
            const Terminal::color & c )
```

Definition at line 66 of file term.cpp.

```
00066                                                                   {
00067     char buf[6];
00068     sprintf(buf,"%d",Terminal::colormap.find(c)->second);
00069     os « "\e[1;" « buf « "m";
00070     return os;
00071 }
```

References Markov::API::CLI::Terminal::colormap.

9.54 term.cpp

```
00001 /** @file term.cpp
00002  * @brief Terminal handler for pretty stuff like colors
00003  * @authors Ata Hakçıl
00004  *
```

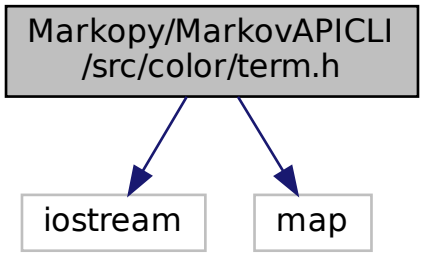
```
00005  * @copydoc Markov::API::CLI::Terminal
00006  */
00007
00008 #include "term.h"
00009 #include <string>
00010
00011 using namespace Markov::API::CLI;
00012
00013 //Windows text processing is different from unix systems, so use windows header and text attributes
00014 #ifdef _WIN32
00015
00016 HANDLE Terminal::_stdout;
00017 HANDLE Terminal::_stderr;
00018
00019 std::map<Terminal::color, DWORD> Terminal::colormap = {
00020     {Terminal::color::BLACK, 0},
00021     {Terminal::color::BLUE, 1},
00022     {Terminal::color::GREEN, 2},
00023     {Terminal::color::CYAN, 3},
00024     {Terminal::color::RED, 4},
00025     {Terminal::color::MAGENTA, 5},
00026     {Terminal::color::BROWN, 6},
00027     {Terminal::color::LIGHTGRAY, 7},
00028     {Terminal::color::DARKGRAY, 8},
00029     {Terminal::color::YELLOW, 14},
00030     {Terminal::color::WHITE, 15},
00031     {Terminal::color::RESET, 15},
00032 };
00033
00034
00035 Terminal::Terminal() {
00036     Terminal::_stdout = GetStdHandle(STD_OUTPUT_HANDLE);
00037     Terminal::_stderr = GetStdHandle(STD_ERROR_HANDLE);
00038 }
00039
00040 std::ostream& operator«(std::ostream& os, const Terminal::color& c) {
00041     SetConsoleTextAttribute(Terminal::_stdout, Terminal::colormap.find(c)->second);
00042     return os;
00043 }
00044
00045 #else
00046
00047 std::map<Terminal::color, int> Terminal::colormap = {
00048     {Terminal::color::BLACK, 30},
00049     {Terminal::color::BLUE, 34},
00050     {Terminal::color::GREEN, 32},
00051     {Terminal::color::CYAN, 36},
00052     {Terminal::color::RED, 31},
00053     {Terminal::color::MAGENTA, 35},
00054     {Terminal::color::BROWN, 0},
00055     {Terminal::color::LIGHTGRAY, 0},
00056     {Terminal::color::DARKGRAY, 0},
00057     {Terminal::color::YELLOW, 33},
00058     {Terminal::color::WHITE, 37},
00059     {Terminal::color::RESET, 0},
00060 };
00061
00062 Terminal::Terminal() {
00063     /*this->;*/
00064 }
00065
00066 std::ostream& operator«(std::ostream& os, const Terminal::color& c) {
00067     char buf[6];
00068     sprintf(buf,"%d",Terminal::colormap.find(c)->second);
00069     os « "\e[1;" « buf « "m";
00070     return os;
00071 }
00072
00073
00074
00075
00076 #endif
```

9.55 Markopy/MarkovAPICLI/src/color/term.h File Reference

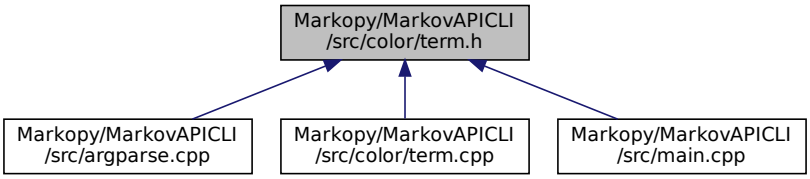
Terminal handler for pretty stuff like colors.

```
#include <iostream>
#include <map>
```

Include dependency graph for term.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::API::CLI::Terminal

  pretty colors for Terminal. Windows Only.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::API

  Namespace for the MarkovPasswords API.
- Markov::API::CLI

  Structure to hold parsed cli arguements.

Macros

- #define TERM_FAIL "[" << Markov::API::CLI::Terminal::color::RED << "+" << Markov::API::CLI::Terminal::color::RESET << "] "
- #define TERM_INFO "[" << Markov::API::CLI::Terminal::color::BLUE << "+" << Markov::API::CLI::Terminal::color::RESET << "] "
- #define TERM_WARN "[" << Markov::API::CLI::Terminal::color::YELLOW << "+" << Markov::API::CLI::Terminal::color::RESE << "] "
- #define TERM_SUCC "[" << Markov::API::CLI::Terminal::color::GREEN << "+" << Markov::API::CLI::Terminal::color::RESET << "] "

Functions

- std::ostream & Markov::API::CLI::operator<< (std::ostream &os, const Markov::API::CLI::Terminal::color &c)

9.55.1 Detailed Description

Terminal handler for pretty stuff like colors.

Authors

Ata Hakçıl

pretty colors for Terminal. Windows Only.
Definition in file term.h.

9.55.2 Macro Definition Documentation

9.55.2.1 TERM_FAIL

```
#define TERM_FAIL "[" << Markov::API::CLI::Terminal::color::RED << "+" << Markov::API::CLI::Terminal::color
<< "] "
```

Definition at line 17 of file term.h.

9.55.2.2 TERM_INFO

```
#define TERM_INFO "[" << Markov::API::CLI::Terminal::color::BLUE << "+" << Markov::API::CLI::Terminal::colo
<< "] "
```

Definition at line 18 of file term.h.

9.55.2.3 TERM_SUCC

```
#define TERM_SUCC "[" << Markov::API::CLI::Terminal::color::GREEN << "+" << Markov::API::CLI::Terminal::col
<< "] "
```

Definition at line 20 of file term.h.

9.55.2.4 TERM_WARN

```
#define TERM_WARN "[" << Markov::API::CLI::Terminal::color::YELLOW << "+" << Markov::API::CLI::Terminal::co
<< "] "
```

Definition at line 19 of file term.h.

9.56 term.h

```
00001 /** @file term.h
00002  * @brief Terminal handler for pretty stuff like colors
00003  * @authors Ata Hakçıl
00004  *
00005  * @copydoc Markov::API::CLI::Terminal
00006  */
00007
00008 #pragma once
00009
00010 #ifdef _WIN32
00011 #include <Windows.h>
00012 #endif
00013
00014 #include <iostream>
00015 #include <map>
00016
00017 #define TERM_FAIL "[" « Markov::API::CLI::Terminal::color::RED « "+" «
       Markov::API::CLI::Terminal::color::RESET « "] "
```

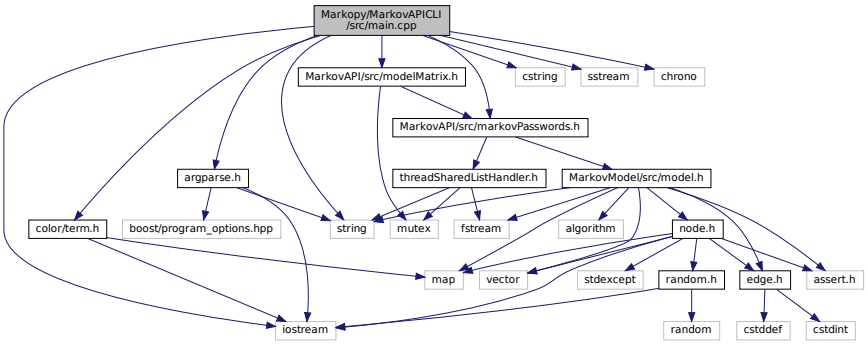
```
00018 #define TERM_INFO "[" « Markov::API::CLI::Terminal::color::BLUE « "+" «
       Markov::API::CLI::Terminal::color::RESET « "] "
00019 #define TERM_WARN "[" « Markov::API::CLI::Terminal::color::YELLOW « "+" «
       Markov::API::CLI::Terminal::color::RESET « "] "
00020 #define TERM_SUCC "[" « Markov::API::CLI::Terminal::color::GREEN « "+" «
       Markov::API::CLI::Terminal::color::RESET « "] "
00021
00022 namespace Markov::API::CLI{
00023     /** @brief pretty colors for Terminal. Windows Only.
00024     */
00025     class Terminal {
00026     public:
00027
00028         /** Default constructor.
00029         * Get references to stdout and stderr handles.
00030         */
00031         Terminal();
00032
00033         enum color { RESET, BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, LIGHTGRAY,
      DARKGRAY, BROWN };
00034         #ifdef _WIN32
00035         static HANDLE _stdout;
00036         static HANDLE _stderr;
00037         static std::map<Markov::API::CLI::Terminal::color, DWORD> colormap;
00038         #else
00039         static std::map<Markov::API::CLI::Terminal::color, int> colormap;
00040         #endif
00041
00042
00043
00044         static std::ostream endl;
00045
00046
00047     };
00048
00049     /** overload for std::cout.
00050     */
00051     std::ostream& operator«(std::ostream& os, const Markov::API::CLI::Terminal::color& c);
00052
00053 }
```

9.57 Markopy/MarkovAPICLI/src/main.cpp File Reference

Test cases for Markov::API::ModelMatrix.

```
#include <iostream>
#include "color/term.h"
#include "argparse.h"
#include <string>
#include <cstring>
#include <sstream>
#include "MarkovAPI/src/markovPasswords.h"
#include "MarkovAPI/src/modelMatrix.h"
#include <chrono>
```

Include dependency graph for main.cpp:



Functions

- int main (int argc, char **argv)

  Launch CLI tool.

9.57.1 Detailed Description

Test cases for Markov::API::ModelMatrix.

Authors

Ata Hakçıl, Celal Sahir Çetiner

Parse command line arguements.
Class to flatten and reduce Markov::Model to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)
To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are sychronized and files are flushed every 50M operations.
Markov::Model with char represented nodes. Includes wrappers for Markov::Model and additional helper functions to handle file I/O
This class is an extension of Markov::Model<char>, with higher level abstractions such as train and generate.
Definition in file main.cpp.

9.57.2 Function Documentation

9.57.2.1 main()

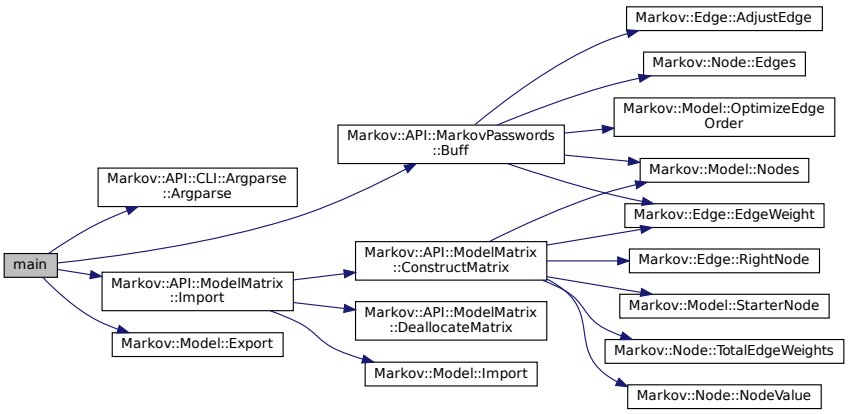
```
int main (
            int argc,
            char ** argv )
```

Launch CLI tool.
Definition at line 23 of file main.cpp.

```
00023                                {
00024
00025     Markov::API::CLI::Terminal t;
00026     /*
00027     ProgramOptions* p  = Argparse::parse(argc, argv);
00028
00029     if (p==0 || p->bFailure) {
00030         std::cout « TERM_FAIL « "Arguments Failed to Parse" « std::endl;
00031         Argparse::help();
00032     }*/
00033     Markov::API::CLI::Argparse a(argc,argv);
00034
00035     Markov::API::ModelMatrix markovPass;
00036     std::cerr « "Importing model.\n";
00037     markovPass.Import("models/finished.mdl");
00038     std::cerr « "Import done. \n";
00039
00040     markovPass.Buff("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~", 100);
00041
00042     markovPass.Export("models/buffed-symbols-100fold.mdl");
00043     //markovPass.ConstructMatrix();
00044     //std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
00045     //markovPass.FastRandomWalk(50000000,"/media/ignis/Stuff/wordlist.txt",6,12,25, true);
00046     //markovPass.FastRandomWalk(500000000,"/media/ignis/Stuff/wordlist2.txt",6,12,25, true);
00047     //std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
00048
00049     //std::cerr « "Finished in:" « std::chrono::duration_cast<std::chrono::milliseconds> (end -
      begin).count() « " milliseconds" « std::endl;
00050     return 0;
00051 }
```

References Markov::API::CLI::Argparse::Argparse(), Markov::API::MarkovPasswords::Buff(), Markov::Model< NodeStorageType >::E and Markov::API::ModelMatrix::Import().

Here is the call graph for this function:



9.58 main.cpp

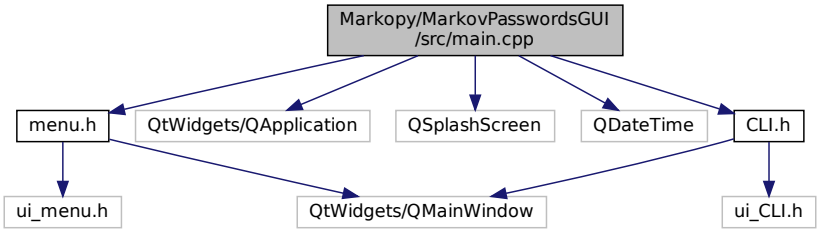
```
00001 /** @file main.cpp
00002  * @brief Test cases for Markov::API::ModelMatrix
00003  * @authors Ata Hakçıl, Celal Sahir Çetiner
00004  *
00005  * @copydoc Markov::API::CLI::Argparse
00006  * @copydoc Markov::API::ModelMatrix
00007  * @copydoc Markov::API::MarkovPasswords
00008  */
00009
00010 #pragma once
00011 #include <iostream>
00012 #include "color/term.h"
00013 #include "argparse.h"
00014 #include <string>
00015 #include <cstring>
00016 #include <sstream>
00017 #include "MarkovAPI/src/markovPasswords.h"
00018 #include "MarkovAPI/src/modelMatrix.h"
00019 #include <chrono>
00020
00021 /** @brief Launch CLI tool.
00022 */
00023 int main(int argc, char** argv) {
00024
00025     Markov::API::CLI::Terminal t;
00026     /*
00027     ProgramOptions* p  = Argparse::parse(argc, argv);
00028
00029     if (p==0 || p->bFailure) {
00030         std::cout « TERM_FAIL « "Arguments Failed to Parse" « std::endl;
00031         Argparse::help();
00032     }*/
00033     Markov::API::CLI::Argparse a(argc,argv);
00034
00035     Markov::API::ModelMatrix markovPass;
00036     std::cerr « "Importing model.\n";
00037     markovPass.Import("models/finished.mdl");
00038     std::cerr « "Import done. \n";
00039
00040     markovPass.Buff("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~", 100);
00041
00042     markovPass.Export("models/buffed-symbols-100fold.mdl");
00043     //markovPass.ConstructMatrix();
00044     //std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
00045     //markovPass.FastRandomWalk(50000000,"/media/ignis/Stuff/wordlist.txt",6,12,25, true);
00046     //markovPass.FastRandomWalk(500000000,"/media/ignis/Stuff/wordlist2.txt",6,12,25, true);
00047     //std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
00048
00049     //std::cerr « "Finished in:" « std::chrono::duration_cast<std::chrono::milliseconds> (end -
      begin).count() « " milliseconds" « std::endl;
00050     return 0;
00051 }
```

9.59 Markopy/MarkovPasswordsGUI/src/main.cpp File Reference

Entry point for GUI.

```
#include "menu.h"
#include <QtWidgets/QApplication>
#include <QSplashScreen>
#include <QDateTime>
#include "CLI.h"
```

Include dependency graph for main.cpp:



Functions

- int main (int argc, char ∗argv[ ])

  Launch UI.

9.59.1 Detailed Description

Entry point for GUI.

Authors

Yunus Emre Yılmaz

Definition in file main.cpp.

9.59.2 Function Documentation

9.59.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Launch UI.

Definition at line 18 of file main.cpp.

```
00019 {
00020
00021
00022
00023     QApplication a(argc, argv);
00024
00025     QPixmap loadingPix("views/startup.jpg");
00026     QSplashScreen splash(loadingPix);
00027     splash.show();
00028     QDateTime time = QDateTime::currentDateTime();
00029     QDateTime currentTime = QDateTime::currentDateTime();   //Record current time
00030     while (time.secsTo(currentTime) <= 5)                   //5 is the number of seconds to delay
00031     {
00032         currentTime = QDateTime::currentDateTime();
00033         a.processEvents();
00034     };
```

```
00035
00036
00037     CLI w;
00038     w.show();
00039     splash.finish(&w);
00040     return a.exec();
00041 }
```

9.60 main.cpp

```
00001 /** @file main.cpp
00002  * @brief Entry point for GUI
00003  * @authors Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 //#include "MarkovPasswordsGUI.h"
00008 #include "menu.h"
00009 #include <QtWidgets/QApplication>
00010 #include <QSplashScreen>
00011 #include < QDateTime >
00012 #include "CLI.h"
00013
00014 using namespace Markov::GUI;
00015
00016 /** @brief Launch UI.
00017 */
00018 int main(int argc, char *argv[])
00019 {
00020
00021
00022
00023     QApplication a(argc, argv);
00024
00025     QPixmap loadingPix("views/startup.jpg");
00026     QSplashScreen splash(loadingPix);
00027     splash.show();
00028     QDateTime time = QDateTime::currentDateTime();
00029     QDateTime currentTime = QDateTime::currentDateTime();   //Record current time
00030     while (time.secsTo(currentTime) <= 5)                   //5 is the number of seconds to delay
00031     {
00032         currentTime = QDateTime::currentDateTime();
00033         a.processEvents();
00034     };
00035
00036
00037     CLI w;
00038     w.show();
00039     splash.finish(&w);
00040     return a.exec();
00041 }
```

9.61 Markopy/MarkovAPICLI/src/scripts/model_2gram.py File Reference

Namespaces

- model_2gram

Variables

- model_2gram.alphabet = string.printable

  password alphabet
- model_2gram.f = open('../../models/2gram.mdl', "wb")

  output file handle

9.62 model_2gram.py

```
00001 #!/usr/bin/python3
00002 """
00003   python script for generating a 2gram model
00004 """
00005
00006 import string
00007 import re
```

```
00008
00009
00010 alphabet = string.printable
00011 alphabet = re.sub('\s', '', alphabet)
00012 print(f"alphabet={alphabet}")
00013 #exit()
00014
00015
00016 f = open('../../models/2gram.mdl', "wb")
00017 #tie start nodes
00018 for sym in alphabet:
00019     f.write(b"\x00,1," + bytes(sym, encoding='ascii') + b"\n")
00020
00021 #tie terminator nodes
00022 for sym in alphabet:
00023     f.write(bytes(sym, encoding='ascii')+ b",1,\xff\n")
00024
00025 #tie internals
00026 for src in alphabet:
00027     for target in alphabet:
00028         f.write(bytes(src, encoding='ascii') + b",1," + bytes(target, encoding='ascii') + b"\n")
```

9.63 Markopy/MarkovAPICLI/src/scripts/random_model.py File Reference

Namespaces

- random_model

Variables

- random_model.alphabet = string.printable

  password alphabet

- random_model.f = open('../../models/random.mdl', "wb")

  output file handle

9.64 random_model.py

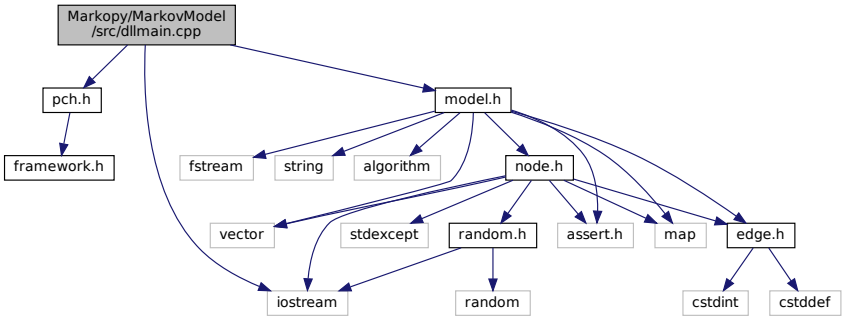
```
00001 #!/usr/bin/python3
00002 """
00003   python script for generating a 2gram model
00004 """
00005
00006 import string
00007 import re
00008
00009
00010 alphabet = string.printable
00011 alphabet = re.sub('\s', '', alphabet)
00012 print(f"alphabet={alphabet}")
00013 #exit()
00014
00015
00016 f = open('../../models/random.mdl', "wb")
00017 #tie start nodes
00018 for sym in alphabet:
00019     f.write(b"\x00,1," + bytes(sym, encoding='ascii') + b"\n")
00020
00021 #tie terminator nodes
00022 for sym in alphabet:
00023     f.write(bytes(sym, encoding='ascii')+ b",1,\xff\n")
00024
00025 #tie internals
00026 for src in alphabet:
00027     for target in alphabet:
00028         f.write(bytes(src, encoding='ascii') + b",1," + bytes(target, encoding='ascii') + b"\n")
```

9.65 Markopy/MarkovModel/src/dllmain.cpp File Reference

DLLMain for dynamic windows library.

```
#include "pch.h"
#include "model.h"
#include <iostream>
```

Include dependency graph for dllmain.cpp:



9.65.1 Detailed Description

DLLMain for dynamic windows library.

Authors

Ata Hakçıl

class for the final Markov Model, constructed from nodes and edges. Each atomic piece of the generation result is stored in a node, while edges contain the relation weights. *Extending:* To extend the class, implement the template and inherit from it, as "class MyModel : public Markov::Model<char>". For a complete demonstration of how to extend the class, see MarkovPasswords.
Whole model can be defined as a list of the edges, as dangling nodes are pointless. This approach is used for the import/export operations. For more information on importing/exporting model, check out the github readme and wiki page.
Definition in file dllmain.cpp.

9.66 dllmain.cpp

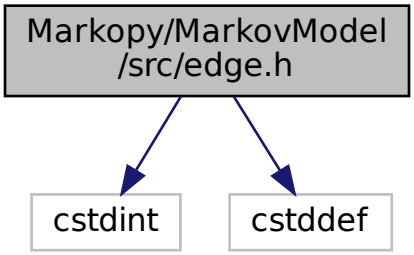
```
00001 /** @file dllmain.cpp
00002  * @brief DLLMain for dynamic windows library
00003  * @authors Ata Hakçıl
00004  *
00005  * @copydoc Markov::Model
00006  */
00007
00008 #include "pch.h"
00009 #include "model.h"
00010 #include <iostream>
00011
00012
00013 #ifdef _WIN32
00014 __declspec(dllexport) void dll_loadtest() {
00015     std::cout « "External function called.\n";
00016     //cudaTestEntry();
00017 }
00018
00019 BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)
00020 {
00021     switch (ul_reason_for_call)
00022     {
00023     case DLL_PROCESS_ATTACH:
00024     case DLL_THREAD_ATTACH:
00025     case DLL_THREAD_DETACH:
00026     case DLL_PROCESS_DETACH:
00027         break;
00028     }
00029     return TRUE;
00030 }
00031
00032 #endif
```

9.67 Markopy/MarkovModel/src/edge.h File Reference

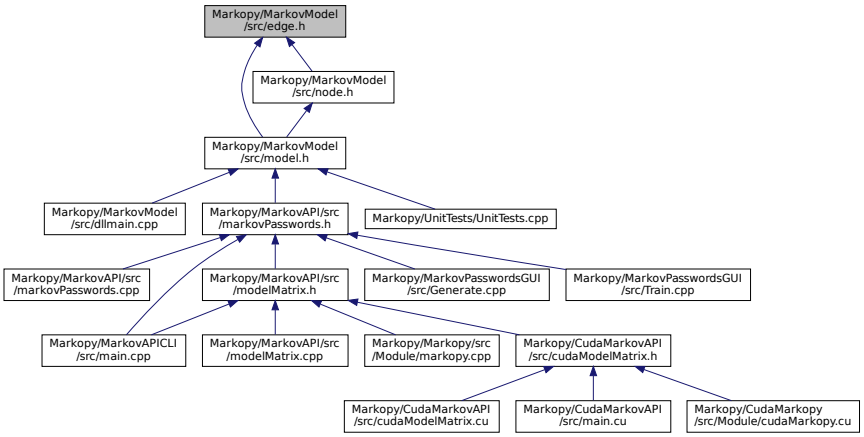
Edge class template.

```
#include <cstdint>
#include <cstddef>
```

Include dependency graph for edge.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::Edge< NodeStorageType >

  Edge class used to link nodes in the model together.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

9.67.1 Detailed Description

Edge class template.

Authors

Ata Hakçıl, Osman Ömer Yıldıztugay

Edge class used to link nodes in the model together. Has LeftNode, RightNode, and EdgeWeight of the edge. Edges are *UNIDIRECTIONAL* in this model. They can only be traversed LeftNode to RightNode.
Definition in file edge.h.

9.68 edge.h

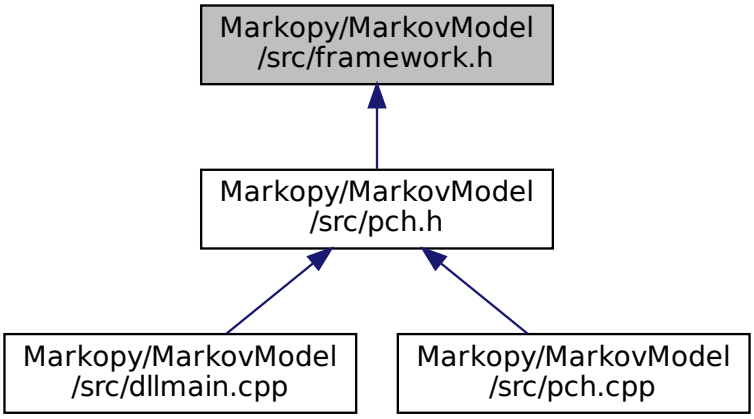
```
00001 /** @file edge.h
00002  * @brief Edge class template
00003  * @authors Ata Hakçıl, Osman Ömer Yıldıztugay
00004  *
00005  * @copydoc Markov::Edge
00006  */
00007
00008 #pragma once
00009 #include <cstdint>
00010 #include <cstddef>
00011
00012 namespace Markov {
00013
00014     template <typename NodeStorageType>
00015     class Node;
00016
00017     /** @brief Edge class used to link nodes in the model together.
00018     *
00019     Has LeftNode, RightNode, and EdgeWeight of the edge.
00020     Edges are *UNIDIRECTIONAL* in this model. They can only be traversed LeftNode to RightNode.
00021     */
00022     template <typename NodeStorageType>
00023     class Edge {
00024     public:
00025
00026         /** @brief Default constructor.
00027         */
00028         Edge<NodeStorageType>();
00029
00030         /** @brief Constructor. Initialize edge with given RightNode and LeftNode
00031          * @param _left - Left node of this edge.
00032          * @param _right - Right node of this edge.
00033          *
00034          * @b Example @b Use: Construct edge
00035          * @code{.cpp}
00036          * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00037          * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00038          * Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00039          * @endcode
00040          *
00041         */
00042         Edge<NodeStorageType>(Node<NodeStorageType>* _left, Node<NodeStorageType>* _right);
00043
00044         /** @brief Adjust the edge EdgeWeight with offset.
00045          * Adds the offset parameter to the edge EdgeWeight.
00046          * @param offset - NodeValue to be added to the EdgeWeight
00047          *
00048          * @b Example @b Use: Construct edge
00049          * @code{.cpp}
00050          * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00051          * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00052          * Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00053          *
00054          * e1->AdjustEdge(25);
00055          *
00056          * @endcode
00057         */
00058         void AdjustEdge(long int offset);
00059
00060         /** @brief Traverse this edge to RightNode.
00061          * @return Right node. If this is a terminator node, return NULL
00062          *
00063          *
00064          * @b Example @b Use: Traverse a node
00065          * @code{.cpp}
00066          * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00067          * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00068          * Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00069          *
00070          * e1->AdjustEdge(25);
00071          * Markov::Edge<unsigned char>* e2 = e1->traverseNode();
00072          * @endcode
00073          *
```

```
00074         */
00075         inline Node<NodeStorageType>* TraverseNode();
00076
00077         /** @brief Set LeftNode of this edge.
00078         * @param node - Node to be linked with.
00079         */
00080         void SetLeftEdge (Node<NodeStorageType>*);
00081         /** @brief Set RightNode of this edge.
00082         * @param node - Node to be linked with.
00083         */
00084         void SetRightEdge(Node<NodeStorageType>*);
00085
00086         /** @brief return edge's EdgeWeight.
00087         * @return edge's EdgeWeight.
00088         */
00089         inline uint64_t EdgeWeight();
00090
00091         /** @brief return edge's LeftNode
00092         * @return edge's LeftNode.
00093         */
00094         Node<NodeStorageType>* LeftNode();
00095
00096         /** @brief return edge's RightNode
00097         * @return edge's RightNode.
00098         */
00099         inline Node<NodeStorageType>* RightNode();
00100
00101     private:
00102         /**
00103             @brief source node
00104         */
00105         Node<NodeStorageType>* _left;
00106
00107         /**
00108             @brief target node
00109         */
00110         Node<NodeStorageType>* _right;
00111
00112         /** @brief
00113             Edge Edge Weight
00114         */
00115         long int _weight;
00116     };
00117
00118
00119 };
00120
00121 //default constructor of edge
00122 template <typename NodeStorageType>
00123 Markov::Edge<NodeStorageType>::Edge() {
00124     this->_left = NULL;
00125     this->_right = NULL;
00126     this->_weight = 0;
00127 }
00128 //constructor of edge
00129 template <typename NodeStorageType>
00130 Markov::Edge<NodeStorageType>::Edge(Markov::Node<NodeStorageType>* _left,
       Markov::Node<NodeStorageType>* _right) {
00131     this->_left = _left;
00132     this->_right = _right;
00133     this->_weight = 0;
00134 }
00135 //to AdjustEdge the edges by the edge with its offset
00136 template <typename NodeStorageType>
00137 void Markov::Edge<NodeStorageType>::AdjustEdge(long int offset) {
00138     this->_weight += offset;
00139     this->LeftNode()->UpdateTotalVerticeWeight(offset);
00140 }
00141 //to TraverseNode the node
00142 template <typename NodeStorageType>
00143 inline Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::TraverseNode() {
00144     if (this->RightNode()->NodeValue() == 0xff) //terminator node
00145         return NULL;
00146     return _right;
00147 }
00148 //to set the LeftNode of the node
00149 template <typename NodeStorageType>
00150 void Markov::Edge<NodeStorageType>::SetLeftEdge(Markov::Node<NodeStorageType>* n) {
00151     this->_left = n;
00152 }
00153 //to set the RightNode of the node
00154 template <typename NodeStorageType>
00155 void Markov::Edge<NodeStorageType>::SetRightEdge(Markov::Node<NodeStorageType>* n) {
00156     this->_right = n;
00157 }
00158 //to get the EdgeWeight of the node
00159 template <typename NodeStorageType>
```

```
00160 inline uint64_t Markov::Edge<NodeStorageType>::EdgeWeight() {
00161     return this->_weight;
00162 }
00163 //to get the LeftNode of the node
00164 template <typename NodeStorageType>
00165 Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::LeftNode() {
00166     return this->_left;
00167 }
00168 //to get the RightNode of the node
00169 template <typename NodeStorageType>
00170 inline Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::RightNode() {
00171     return this->_right;
00172 }
```

9.69 Markopy/MarkovModel/src/framework.h File Reference

for windows dynamic library
This graph shows which files directly or indirectly include this file:



Macros

- #define WIN32_LEAN_AND_MEAN

9.69.1 Detailed Description

for windows dynamic library

Authors

Ata Hakçıl

Definition in file framework.h.

9.69.2 Macro Definition Documentation

9.69.2.1 WIN32_LEAN_AND_MEAN

```
#define WIN32_LEAN_AND_MEAN
```

Definition at line 9 of file framework.h.

9.70 framework.h

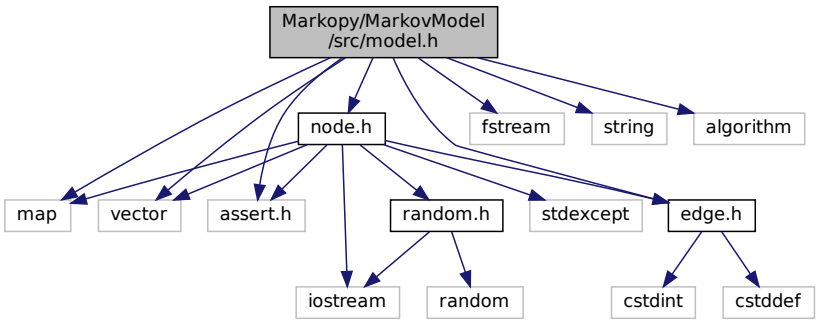
```
00001 /** @file framework.h
00002  * @brief for windows dynamic library
00003  * @authors Ata Hakçıl
00004  *
00005  */
00006
00007 #pragma once
00008
00009 #define WIN32_LEAN_AND_MEAN             // Exclude rarely-used stuff from Windows headers
00010 // Windows Header Files
00011
00012 #ifdef _WIN32
00013 #include <windows.h>
00014 #endif
```

9.71 Markopy/MarkovModel/src/model.h File Reference

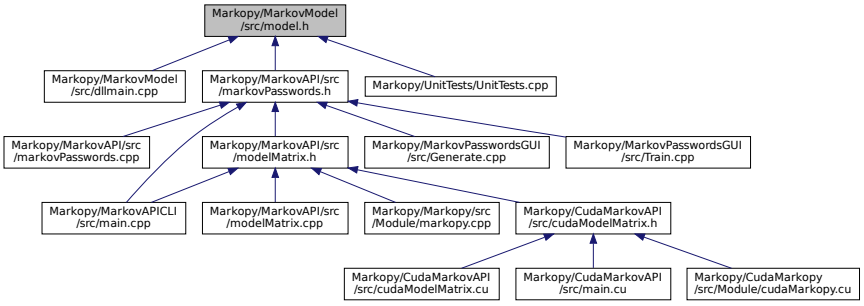
Model class template.

```
#include <map>
#include <vector>
#include <fstream>
#include <assert.h>
#include <string>
#include <algorithm>
#include "node.h"
#include "edge.h"
```

Include dependency graph for model.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::Model< NodeStorageType >

  class for the final Markov Model, constructed from nodes and edges.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

9.71.1 Detailed Description

Model class template.

Authors

Ata Hakçıl, Osman Ömer Yıldıztugay

class for the final Markov Model, constructed from nodes and edges. Each atomic piece of the generation result is stored in a node, while edges contain the relation weights. *Extending:* To extend the class, implement the template and inherit from it, as "class MyModel : public Markov::Model<char>". For a complete demonstration of how to extend the class, see MarkovPasswords.
Whole model can be defined as a list of the edges, as dangling nodes are pointless. This approach is used for the import/export operations. For more information on importing/exporting model, check out the github readme and wiki page.
Definition in file model.h.

9.72 model.h

```
00001 /** @file model.h
00002  * @brief Model class template
00003  * @authors Ata Hakçıl, Osman Ömer Yıldıztugay
00004  *
00005  * @copydoc Markov::Model
00006  */
00007
00008
00009
00010 #pragma once
00011 #include <map>
00012 #include <vector>
00013 #include <fstream>
00014 #include <assert.h>
00015 #include <string>
00016 #include <algorithm>
00017 #include "node.h"
00018 #include "edge.h"
00019
00020 /**
00021     @brief Namespace for the markov-model related classes.
00022     Contains Model, Node and Edge classes
00023 */
00024 namespace Markov {
00025
00026     template <typename NodeStorageType>
00027     class Node;
00028
00029     template <typename NodeStorageType>
00030     class Edge;
00031
00032     template <typename NodeStorageType>
00033
00034     /** @brief class for the final Markov Model, constructed from nodes and edges.
00035      *
00036      * Each atomic piece of the generation result is stored in a node, while edges contain the
       relation weights.
00037      * *Extending:*
00038      * To extend the class, implement the template and inherit from it, as "class MyModel : public
       Markov::Model<char>".
00039      * For a complete demonstration of how to extend the class, see MarkovPasswords.
00040      *
00041      * 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.
00042      * For more information on importing/exporting model, check out the github readme and wiki page.
```

```
00043      *
00044     */
00045     class Model {
00046     public:
00047
00048         /** @brief Initialize a model with only start and end nodes.
00049          *
00050          * Initialize an empty model with only a starterNode
00051          * Starter node is a special kind of node that has constant 0x00 value, and will be used to
      initiate the generation execution from.
00052         */
00053         Model<NodeStorageType>();
00054
00055         /** @brief Do a random walk on this model.
00056          *
00057          * Start from the starter node, on each node, invoke RandomNext using the random engine on
      current node, until terminator node is reached.
00058          * If terminator node is reached before minimum length criateria is reached, ignore the last
      selection and re-invoke randomNext
00059          *
00060          * If maximum length criteria is reached but final node is not, cut off the generation and
      proceed to the final node.
00061          * This function takes Markov::Random::RandomEngine as a parameter to generate pseudo random
      numbers from
00062          *
00063          * This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne
      output is higher in entropy, most use cases
00064          * don't really need super high entropy output, so Markov::Random::Marsaglia is preferable for
      better performance.
00065          *
00066          * This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening
      via maximum length criteria.
00067          *
00068          * @b Example @b Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use
      Marsaglia
00069          * @code{.cpp}
00070          * Markov::Model<char> model;
00071          * Model.import("model.mdl");
00072          * char* res = new char[11];
00073          * Markov::Random::Marsaglia MarsagliaRandomEngine;
00074          * for (int i = 0; i < 10; i++) {
00075          *      this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
00076          *      std::cout « res « "\n";
00077          *  }
00078          * @endcode
00079          *
00080          * @param randomEngine Random Engine to use for the random walks. For examples, see
      Markov::Random::Mersenne and Markov::Random::Marsaglia
00081          * @param minSetting Minimum number of characters to generate
00082          * @param maxSetting Maximum number of character to generate
00083          * @param buffer buffer to write the result to
00084          * @return Null terminated string that was generated.
00085         */
00086         NodeStorageType* RandomWalk(Markov::Random::RandomEngine* randomEngine, int minSetting, int
      maxSetting, NodeStorageType* buffer);
00087
00088         /** @brief Adjust the model with a single string.
00089          *
00090          * Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from
      current node to the next, until NULL character is reached.
00091          *
00092          * Then, update the edge EdgeWeight from current node, to the terminator node.
00093          *
00094          * This function is used for training purposes, as it can be used for adjusting the model with
      each line of the corpus file.
00095          *
00096          * @b Example @b Use: Create an empty model and train it with string: "testdata"
00097          * @code{.cpp}
00098          * Markov::Model<char> model;
00099          * char test[] = "testdata";
00100          * model.AdjustEdge(test, 15);
00101          * @endcode
00102          *
00103          *
00104          * @param string - String that is passed from the training, and will be used to AdjustEdge the
      model with
00105          * @param occurrence - Occurrence of this string.
00106          *
00107          *
00108         */
00109         void AdjustEdge(const NodeStorageType* payload, long int occurrence);
00110
00111         /** @brief Import a file to construct the model.
00112          *
00113          * File contains a list of edges. For more info on the file format, check out the wiki and
      github readme pages.
00114          * Format is: Left_repr;EdgeWeight;right_repr
```

```
00115         *
00116         * Iterate over this list, and construct nodes and edges accordingly.
00117         * @return True if successful, False for incomplete models or corrupt file formats
00118         *
00119         * @b Example @b Use: Import a file from ifstream
00120         * @code{.cpp}
00121         * Markov::Model<char> model;
00122         * std::ifstream file("test.mdl");
00123         * model.Import(&file);
00124         * @endcode
00125         */
00126        bool Import(std::ifstream*);
00127
00128        /** @brief Open a file to import with filename, and call bool Model::Import with std::ifstream
00129         * @return True if successful, False for incomplete models or corrupt file formats
00130         *
00131         * @b Example @b Use: Import a file with filename
00132         * @code{.cpp}
00133         * Markov::Model<char> model;
00134         * model.Import("test.mdl");
00135         * @endcode
00136         */
00137        bool Import(const char* filename);
00138
00139        /** @brief Export a file of the model.
00140         *
00141         * File contains a list of edges.
00142         * Format is: Left_repr;EdgeWeight;right_repr.
00143         * For more information on the format, check out the project wiki or github readme.
00144         *
00145         * Iterate over this vertices, and their edges, and write them to file.
00146         * @return True if successful, False for incomplete models.
00147         *
00148         * @b Example @b Use: Export file to ofstream
00149         * @code{.cpp}
00150         * Markov::Model<char> model;
00151         * std::ofstream file("test.mdl");
00152         * model.Export(&file);
00153         * @endcode
00154         */
00155        bool Export(std::ofstream*);
00156
00157        /** @brief Open a file to export with filename, and call bool Model::Export with std::ofstream
00158         * @return True if successful, False for incomplete models or corrupt file formats
00159         *
00160         * @b Example @b Use: Export file to filename
00161         * @code{.cpp}
00162         * Markov::Model<char> model;
00163         * model.Export("test.mdl");
00164         * @endcode
00165         */
00166        bool Export(const char* filename);
00167
00168        /** @brief Return starter Node
00169         * @return starter node with 00 NodeValue
00170         */
00171        Node<NodeStorageType>* StarterNode(){ return starterNode;}
00172
00173        /** @brief Return a vector of all the edges in the model
00174         * @return vector of edges
00175         */
00176        std::vector<Edge<NodeStorageType>*>* Edges(){ return &edges;}
00177
00178        /** @brief Return starter Node
00179         * @return starter node with 00 NodeValue
00180         */
00181        std::map<NodeStorageType, Node<NodeStorageType>*>* Nodes(){ return &nodes;}
00182
00183        /** @brief Sort edges of all nodes in the model ordered by edge weights
00184         *
00185         */
00186        void OptimizeEdgeOrder();
00187
00188    private:
00189        /**
00190            @brief Map LeftNode is the Nodes NodeValue
00191            * Map RightNode is the node pointer
00192        */
00193        std::map<NodeStorageType, Node<NodeStorageType>*> nodes;
00194
00195        /**
00196            @brief Starter Node of this model.
00197        */
00198        Node<NodeStorageType>* starterNode;
00199
00200
00201        /**
```

```
00202             @brief A list of all edges in this model.
00203         */
00204         std::vector<Edge<NodeStorageType>*> edges;
00205     };
00206
00207 };
00208
00209 template <typename NodeStorageType>
00210 Markov::Model<NodeStorageType>::Model() {
00211     this->starterNode = new Markov::Node<NodeStorageType>(0);
00212     this->nodes.insert({ 0, this->starterNode });
00213 }
00214
00215 template <typename NodeStorageType>
00216 bool Markov::Model<NodeStorageType>::Import(std::ifstream* f) {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout << "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
      int(targetN->NodeValue()) << "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
00263
00264 template <typename NodeStorageType>
00265 void Markov::Model<NodeStorageType>::OptimizeEdgeOrder(){
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
      Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         //  std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00273         //std::cout << "\n";
00274     }
00275     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00276     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00277 }
00278
00279 template <typename NodeStorageType>
00280 bool Markov::Model<NodeStorageType>::Import(const char* filename) {
00281     std::ifstream importfile;
00282     importfile.open(filename);
00283     return this->Import(&importfile);
00284
00285 }
00286
```

```
00287 template <typename NodeStorageType>
00288 bool Markov::Model<NodeStorageType>::Export(std::ofstream* f) {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
      e->RightNode()->NodeValue() << "\n";
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
      "\n";
00294     }
00295
00296     return true;
00297 }
00298
00299 template <typename NodeStorageType>
00300 bool Markov::Model<NodeStorageType>::Export(const char* filename) {
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
00305
00306 template <typename NodeStorageType>
00307 NodeStorageType* Markov::Model<NodeStorageType>::RandomWalk(Markov::Random::RandomEngine*
      randomEngine, int minSetting, int maxSetting, NodeStorageType* buffer) {
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL){
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue();
00327     }
00328
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
00335
00336 template <typename NodeStorageType>
00337 void Markov::Model<NodeStorageType>::AdjustEdge(const NodeStorageType* payload, long int occurrence) {
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

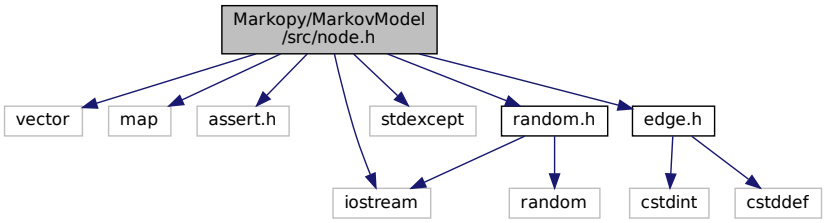
9.73 Markopy/MarkovModel/src/node.h File Reference

Node class template.

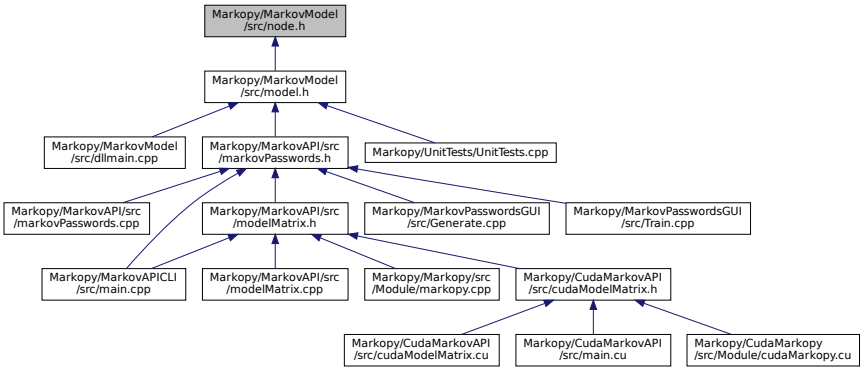
```
#include <vector>
#include <map>
#include <assert.h>
#include <iostream>
#include <stdexcept>
```

```
#include "edge.h"
#include "random.h"
```

Include dependency graph for node.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::Node< storageType >

  A node class that for the vertices of model. Connected with eachother using Edge.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

9.73.1 Detailed Description

Node class template.

Authors

Ata Hakçıl, Osman Ömer Yıldıztugay

A node class that for the vertices of model. Connected with eachother using Edge. This class will later be templated to accept other data types than char∗.
Definition in file node.h.

9.74 node.h

```
00001 /** @file node.h
```

```
00002  * @brief Node class template
00003  * @authors Ata Hakçıl, Osman Ömer Yıldıztugay
00004  *
00005  * @copydoc Markov::Node
00006  */
00007
00008
00009 #pragma once
00010 #include <vector>
00011 #include <map>
00012 #include <assert.h>
00013 #include <iostream>
00014 #include <stdexcept> // To use runtime_error
00015 #include "edge.h"
00016 #include "random.h"
00017 namespace Markov {
00018
00019     /** @brief A node class that for the vertices of model. Connected with eachother using Edge
00020     *
00021     * This class will later be templated to accept other data types than char*.
00022     */
00023     template <typename storageType>
00024     class Node {
00025     public:
00026
00027         /** @brief Default constructor. Creates an empty Node.
00028         */
00029         Node<storageType>();
00030
00031         /** @brief Constructor. Creates a Node with no edges and with given NodeValue.
00032          * @param _value - Nodes character representation.
00033          *
00034          * @b Example @b Use: Construct nodes
00035          * @code{.cpp}
00036          * Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00037          * Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00038          * @endcode
00039         */
00040         Node<storageType>(storageType _value);
00041
00042         /** @brief Link this node with another, with this node as its source.
00043          *
00044          * Creates a new Edge.
00045          * @param target - Target node which will be the RightNode() of new edge.
00046          * @return A new node with LeftNode as this, and RightNode as parameter target.
00047          *
00048          * @b Example @b Use: Construct nodes
00049          * @code{.cpp}
00050          * Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00051          * Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00052          * Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
00053          * @endcode
00054         */
00055         Edge<storageType>* Link(Node<storageType>*);
00056
00057         /** @brief Link this node with another, with this node as its source.
00058          *
00059          * *DOES NOT* create a new Edge.
00060          * @param Edge - Edge that will accept this node as its LeftNode.
00061          * @return the same edge as parameter target.
00062          *
00063          * @b Example @b Use: Construct and link nodes
00064          * @code{.cpp}
00065          * Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00066          * Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00067          * Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
00068          * LeftNode->Link(e);
00069          * @endcode
00070         */
00071         Edge<storageType>* Link(Edge<storageType>*);
00072
00073         /** @brief Chose a random node from the list of edges, with regards to its EdgeWeight, and
    TraverseNode to that.
00074          *
00075          * 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.
00076          * At each step, EdgeWeight of the edge is subtracted from the random number, and once it is
    0, next node is selected.
00077          * @return Node that was chosen at EdgeWeight biased random.
00078          *
00079          * @b Example @b Use: Use randomNext to do a random walk on the model
00080          * @code{.cpp}
00081          *  char* buffer[64];
00082          *  Markov::Model<char> model;
00083          *  model.Import("model.mdl");
00084          *  Markov::Node<char>* n = model.starterNode;
00085          *  int len = 0;
```

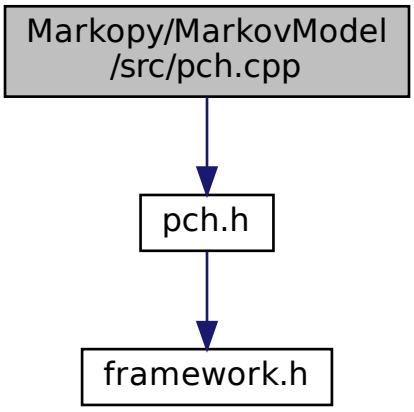
```
00086          *  Markov::Node<char>* temp_node;
00087          *  while (true) {
00088          *      temp_node = n->RandomNext(randomEngine);
00089          *      if (len >= maxSetting) {
00090          *          break;
00091          *      }
00092          *      else if ((temp_node == NULL) && (len < minSetting)) {
00093          *          continue;
00094          *      }
00095          *
00096          *      else if (temp_node == NULL){
00097          *          break;
00098          *      }
00099          *
00100          *      n = temp_node;
00101          *
00102          *      buffer[len++] = n->NodeValue();
00103          *  }
00104          * @endcode
00105         */
00106         Node<storageType>* RandomNext(Markov::Random::RandomEngine* randomEngine);
00107
00108         /** @brief Insert a new edge to the this.edges.
00109          * @param edge - New edge that will be inserted.
00110          * @return true if insertion was successful, false if it fails.
00111          *
00112          * @b Example @b Use: Construct and update edges
00113          *
00114          * @code{.cpp}
00115          * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00116          * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00117          * Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00118          * Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00119          * Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00120          * e1->AdjustEdge(25);
00121          * src->UpdateEdges(e1);
00122          * e2->AdjustEdge(30);
00123          * src->UpdateEdges(e2);
00124          * @endcode
00125         */
00126         bool UpdateEdges(Edge<storageType>*);
00127
00128         /** @brief Find an edge with its character representation.
00129          * @param repr - character NodeValue of the target node.
00130          * @return Edge that is connected between this node, and the target node.
00131          *
00132          * @b Example @b Use: Construct and update edges
00133          *
00134          * @code{.cpp}
00135          * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00136          * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00137          * Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00138          * Markov::Edge<unsigned char>* res = NULL;
00139          * src->Link(target1);
00140          * src->Link(target2);
00141          * res = src->FindEdge('b');
00142          *
00143          * @endcode
00144          *
00145         */
00146         Edge<storageType>* FindEdge(storageType repr);
00147
00148         /** @brief Find an edge with its pointer. Avoid unless neccessary because comptutational cost
       of find by character is cheaper (because of std::map)
00149         * @param target - target node.
00150         * @return Edge that is connected between this node, and the target node.
00151         */
00152         Edge<storageType>* FindEdge(Node<storageType>* target);
00153
00154         /** @brief Return character representation of this node.
00155         * @return character representation at _value.
00156         */
00157         inline unsigned char NodeValue();
00158
00159         /** @brief Change total weights with offset
00160          * @param offset to adjust the vertice weight with
00161         */
00162         void UpdateTotalVerticeWeight(long int offset);
00163
00164         /** @brief return edges
00165         */
00166         inline std::map<storageType, Edge<storageType>*>* Edges();
00167
00168         /** @brief return total edge weights
00169         */
00170         inline long int TotalEdgeWeights();
00171
```

```
00172
00173         std::vector<Edge<storageType>*> edgesV;
00174     private:
00175
00176         /**
00177             @brief Character representation of this node. 0 for starter, 0xff for terminator.
00178         */
00179         storageType _value;
00180
00181         /**
00182             @brief Total weights of the vertices, required by RandomNext
00183         */
00184         long int total_edge_weights;
00185
00186         /**
00187             @brief A map of all edges connected to this node, where this node is at the LeftNode.
00188             * Map is indexed by unsigned char, which is the character representation of the node.
00189         */
00190         std::map<storageType, Edge<storageType>*> edges;
00191     };
00192 };
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202 template <typename storageType>
00203 Markov::Node<storageType>::Node(storageType _value) {
00204     this->_value = _value;
00205     this->total_edge_weights = 0L;
00206 };
00207
00208 template <typename storageType>
00209 Markov::Node<storageType>::Node() {
00210     this->_value = 0;
00211     this->total_edge_weights = 0L;
00212 };
00213
00214 template <typename storageType>
00215 inline unsigned char Markov::Node<storageType>::NodeValue() {
00216     return _value;
00217 }
00218
00219 template <typename storageType>
00220 Markov::Edge<storageType>* Markov::Node<storageType>::Link(Markov::Node<storageType>* n) {
00221     Markov::Edge<storageType>* v = new Markov::Edge<storageType>(this, n);
00222     this->UpdateEdges(v);
00223     return v;
00224 }
00225
00226 template <typename storageType>
00227 Markov::Edge<storageType>* Markov::Node<storageType>::Link(Markov::Edge<storageType>* v) {
00228     v->SetLeftEdge(this);
00229     this->UpdateEdges(v);
00230     return v;
00231 }
00232
00233 template <typename storageType>
00234 Markov::Node<storageType>* Markov::Node<storageType>::RandomNext(Markov::Random::RandomEngine*
       randomEngine) {
00235
00236     //get a random NodeValue in range of total_vertice_weight
00237     long int selection = randomEngine->random() %
       this->total_edge_weights;//distribution()(generator());// distribution(generator);
00238     //make absolute, no negative modulus values wanted
00239     //selection = (selection >= 0) ? selection : (selection + this->total_edge_weights);
00240     for(int i=0;i<this->edgesV.size();i++){
00241         selection -= this->edgesV[i]->EdgeWeight();
00242         if (selection < 0) return this->edgesV[i]->TraverseNode();
00243     }
00244
00245     //if this assertion is reached, it means there is an implementation error above
00246     std::cout << "This should never be reached (node failed to walk to next)\n"; //cant assert from
       child thread
00247     assert(true && "This should never be reached (node failed to walk to next)");
00248     return NULL;
00249 }
00250
00251 template <typename storageType>
00252 bool Markov::Node<storageType>::UpdateEdges(Markov::Edge<storageType>* v) {
00253     this->edges.insert({ v->RightNode()->NodeValue(), v });
00254     this->edgesV.push_back(v);
00255     //this->total_edge_weights += v->EdgeWeight();
```

```
00256     return v->TraverseNode();
00257 }
00258
00259 template <typename storageType>
00260 Markov::Edge<storageType>* Markov::Node<storageType>::FindEdge(storageType repr) {
00261     auto e = this->edges.find(repr);
00262     if (e == this->edges.end()) return NULL;
00263     return e->second;
00264 };
00265
00266 template <typename storageType>
00267 void Markov::Node<storageType>::UpdateTotalVerticeWeight(long int offset) {
00268     this->total_edge_weights += offset;
00269 }
00270
00271 template <typename storageType>
00272 inline std::map<storageType, Markov::Edge<storageType>*>* Markov::Node<storageType>::Edges() {
00273     return &(this->edges);
00274 }
00275
00276 template <typename storageType>
00277 inline long int Markov::Node<storageType>::TotalEdgeWeights() {
00278     return this->total_edge_weights;
00279 }
```

9.75 Markopy/MarkovModel/src/pch.cpp File Reference

For windows dynamic library.
`#include "pch.h"`
Include dependency graph for pch.cpp:



9.75.1 Detailed Description

For windows dynamic library.

Authors

Ata Hakçıl

Definition in file pch.cpp.

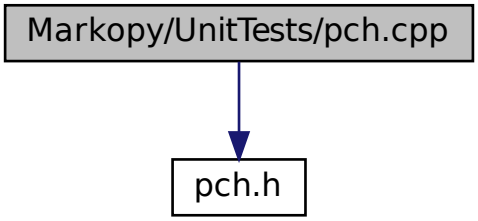
9.76 pch.cpp

```
00001 /** @file pch.cpp
00002  * @brief For windows dynamic library
```

```
00003  * @authors Ata Hakçıl
00004  *
00005  */
00006
00007 // pch.cpp: source file corresponding to the pre-compiled header
00008
00009 #include "pch.h"
00010
00011 // When you are using pre-compiled headers, this source file is necessary for compilation to succeed.
```

9.77 Markopy/UnitTests/pch.cpp File Reference

For windows dynamic library.
`#include "pch.h"`
Include dependency graph for pch.cpp:



9.77.1 Detailed Description

For windows dynamic library.

Authors

Ata Hakçıl

Definition in file pch.cpp.

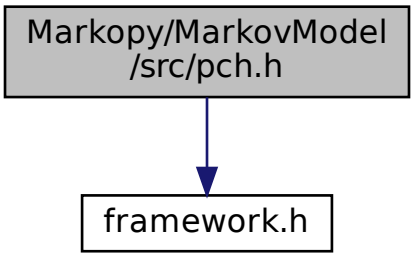
9.78 pch.cpp

```
00001 /** @file pch.cpp
00002  * @brief For windows dynamic library
00003  * @authors Ata Hakçıl
00004  *
00005  */
00006 // pch.cpp: source file corresponding to the pre-compiled header
00007
00008 #include "pch.h"
00009
00010 // When you are using pre-compiled headers, this source file is necessary for compilation to succeed.
```

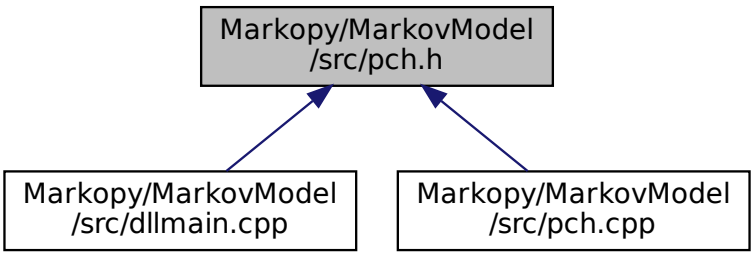
9.79 Markopy/MarkovModel/src/pch.h File Reference

For windows dynamic library.

`#include "framework.h"`
Include dependency graph for pch.h:



This graph shows which files directly or indirectly include this file:



9.79.1 Detailed Description

For windows dynamic library.

Authors

Ata Hakçıl

Definition in file pch.h.

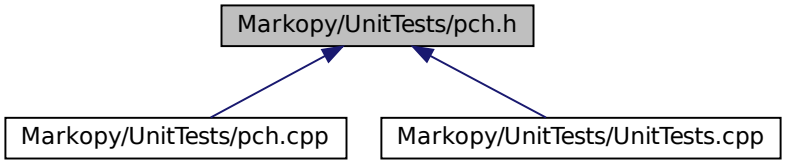
9.80 pch.h

```
00001 /** @file pch.h
00002  * @brief For windows dynamic library
00003  * @authors Ata Hakçıl
00004  *
00005  */
00006 // pch.h: This is a precompiled header file.
00007 // Files listed below are compiled only once, improving build performance for future builds.
00008 // This also affects IntelliSense performance, including code completion and many code browsing
       features.
00009 // However, files listed here are ALL re-compiled if any one of them is updated between builds.
00010 // Do not add files here that you will be updating frequently as this negates the performance
       advantage.
00011
00012 #ifndef PCH_H
00013 #define PCH_H
```

```
00014
00015 // add headers that you want to pre-compile here
00016 #include "framework.h"
00017
00018 #endif //PCH_H
```

9.81 Markopy/UnitTests/pch.h File Reference

For windows dynamic library.
This graph shows which files directly or indirectly include this file:



9.81.1 Detailed Description

For windows dynamic library.

Authors

Ata Hakçıl

Definition in file pch.h.

9.82 pch.h

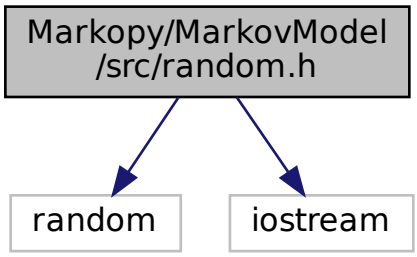
```
00001 /** @file pch.h
00002  * @brief For windows dynamic library
00003  * @authors Ata Hakçıl
00004  *
00005  */
00006 // pch.h: This is a precompiled header file.
00007 // Files listed below are compiled only once, improving build performance for future builds.
00008 // This also affects IntelliSense performance, including code completion and many code browsing
       features.
00009 // However, files listed here are ALL re-compiled if any one of them is updated between builds.
00010 // Do not add files here that you will be updating frequently as this negates the performance
       advantage.
00011
00012 #ifndef PCH_H
00013 #define PCH_H
00014
00015 // add headers that you want to pre-compile here
00016
00017 #endif //PCH_H
```

9.83 Markopy/MarkovModel/src/random.h File Reference

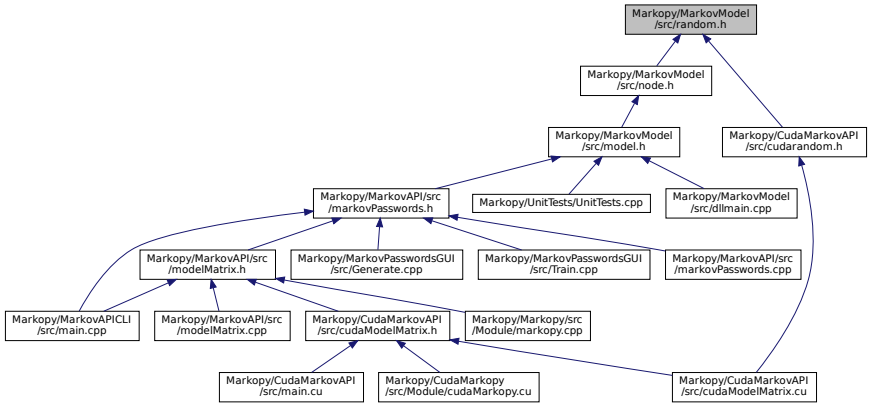
Random engine implementations for Markov.

```
#include <random>
#include <iostream>
```

Include dependency graph for random.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::Random::RandomEngine

  An abstract class for Random Engine.
- class Markov::Random::DefaultRandomEngine

  Implementation using Random.h default random engine.
- class Markov::Random::Marsaglia

  Implementation of Marsaglia Random Engine.
- class Markov::Random::Mersenne

  Implementation of Mersenne Twister Engine.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::Random

  Objects related to RNG.

9.83.1 Detailed Description

Random engine implementations for Markov.

Authors

Ata Hakçıl

An abstract class for Random Engine. This class is used for generating random numbers, which are used for random walking on the graph.
Main reason behind allowing different random engines is that some use cases may favor performance, while some favor good random.
Mersenne can be used for truer random, while Marsaglia can be used for deterministic but fast random.
Implementation using Random.h default random engine. This engine is also used by other engines for seeding.
Example Use: Using Default Engine with RandomWalk

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::DefaultRandomEngine randomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&randomEngine, 5, 10, res);
    std::cout « res « "\n";
 }
```

Example Use: Generating a random number with Marsaglia Engine

```
Markov::Random::DefaultRandomEngine de;
std::cout « de.random();
```

Implementation of Marsaglia Random Engine. This is an implementation of Marsaglia Random engine, which for most use cases is a better fit than other solutions. Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.
This implementation of the Marsaglia Engine is seeded by random.h default random engine. RandomEngine is only seeded once so its not a performance issue.
Example Use: Using Marsaglia Engine with RandomWalk

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout « res « "\n";
 }
```

Example Use: Generating a random number with Marsaglia Engine

```
Markov::Random::Marsaglia me;
std::cout « me.random();
```

Definition in file random.h.

9.84 random.h

```
00001
00002 /** @file random.h
00003  * @brief Random engine implementations for Markov
00004  * @authors Ata Hakçıl
00005  *
00006  * @copydoc Markov::Random::RandomEngine
00007  * @copydoc Markov::Random::DefaultRandomEngine
00008  * @copydoc Markov::Random::Marsaglia
00009  */
00010
00011 #pragma once
00012 #include <random>
00013 #include <iostream>
00014
00015 /**
00016     @brief Objects related to RNG
00017 */
00018 namespace Markov::Random{
00019
00020     /** @brief An abstract class for Random Engine
00021      *
00022      * This class is used for generating random numbers, which are used for random walking on the
      graph.
00023      *
00024      * Main reason behind allowing different random engines is that some use cases may favor
      performance,
00025      * while some favor good random.
00026      *
```

```
00027     * Mersenne can be used for truer random, while Marsaglia can be used for deterministic but fast
      random.
00028     *
00029     */
00030     class RandomEngine{
00031     public:
00032         virtual inline unsigned long random() = 0;
00033     };
00034
00035
00036
00037     /** @brief Implementation using Random.h default random engine
00038      *
00039      * This engine is also used by other engines for seeding.
00040      *
00041      *
00042      * @b Example @b Use: Using Default Engine with RandomWalk
00043      * @code{.cpp}
00044      * Markov::Model<char> model;
00045      * Model.import("model.mdl");
00046      * char* res = new char[11];
00047      * Markov::Random::DefaultRandomEngine randomEngine;
00048      * for (int i = 0; i < 10; i++) {
00049      *      this->RandomWalk(&randomEngine, 5, 10, res);
00050      *      std::cout « res « "\n";
00051      *  }
00052      * @endcode
00053      *
00054      * @b Example @b Use: Generating a random number with Marsaglia Engine
00055      * @code{.cpp}
00056      * Markov::Random::DefaultRandomEngine de;
00057      * std::cout « de.random();
00058      * @endcode
00059      *
00060     */
00061     class DefaultRandomEngine : public RandomEngine{
00062     public:
00063         /** @brief Generate Random Number
00064         * @return random number in long range.
00065         */
00066         inline unsigned long random(){
00067             return this->distribution()(this->generator());
00068         }
00069     protected:
00070
00071         /** @brief Default random device for seeding
00072         *
00073         */
00074         inline std::random_device& rd() {
00075             static std::random_device _rd;
00076             return _rd;
00077         }
00078
00079         /** @brief Default random engine for seeding
00080         *
00081         */
00082         inline std::default_random_engine& generator() {
00083             static std::default_random_engine _generator(rd()());
00084             return _generator;
00085         }
00086
00087         /** @brief Distribution schema for seeding.
00088         *
00089         */
00090         inline std::uniform_int_distribution<long long unsigned>& distribution() {
00091             static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffFFFF);
00092             return _distribution;
00093         }
00094
00095     };
00096
00097
00098     /** @brief Implementation of Marsaglia Random Engine
00099      *
00100      * This is an implementation of Marsaglia Random engine, which for most use cases is a better fit
      than other solutions.
00101      * Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.
00102      *
00103      * This implementation of the Marsaglia Engine is seeded by random.h default random engine.
00104      * RandomEngine is only seeded once so its not a performance issue.
00105      *
00106      * @b Example @b Use: Using Marsaglia Engine with RandomWalk
00107      * @code{.cpp}
00108      * Markov::Model<char> model;
00109      * Model.import("model.mdl");
00110      * char* res = new char[11];
00111      * Markov::Random::Marsaglia MarsagliaRandomEngine;
```

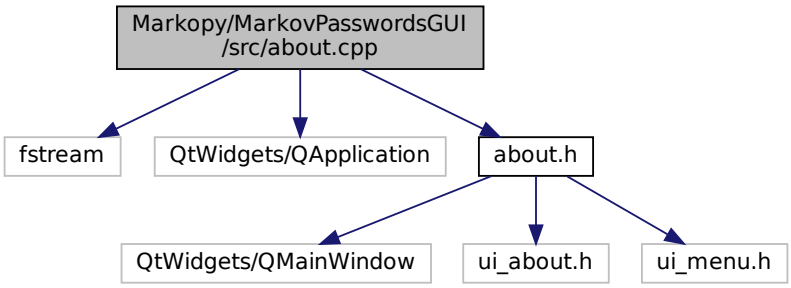
```
00112      * for (int i = 0; i < 10; i++) {
00113      *      this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
00114      *      std::cout « res « "\n";
00115      *  }
00116      * @endcode
00117      *
00118      * @b Example @b Use: Generating a random number with Marsaglia Engine
00119      * @code{.cpp}
00120      * Markov::Random::Marsaglia me;
00121      * std::cout « me.random();
00122      * @endcode
00123      *
00124     */
00125     class Marsaglia : public DefaultRandomEngine{
00126     public:
00127
00128         /** @brief Construct Marsaglia Engine
00129         *
00130         * Initialize x,y and z using the default random engine.
00131         */
00132         Marsaglia(){
00133             this->x = this->distribution()(this->generator());
00134             this->y = this->distribution()(this->generator());
00135             this->z = this->distribution()(this->generator());
00136             //std::cout « "x: " « x « ", y: " « y « ", z: " « z « "\n";
00137         }
00138
00139
00140     inline unsigned long random(){
00141         unsigned long t;
00142         x ^= x « 16;
00143         x ^= x » 5;
00144         x ^= x « 1;
00145
00146         t = x;
00147         x = y;
00148         y = z;
00149         z = t ^ x ^ y;
00150
00151         return z;
00152     }
00153
00154
00155         unsigned long x;
00156         unsigned long y;
00157         unsigned long z;
00158     };
00159
00160
00161     /** @brief Implementation of Mersenne Twister Engine
00162      *
00163      * This is an implementation of Mersenne Twister Engine, which is slow but is a good
    implementation for high entropy pseudorandom.
00164      *
00165      *
00166      * @b Example @b Use: Using Mersenne Engine with RandomWalk
00167      * @code{.cpp}
00168      * Markov::Model<char> model;
00169      * Model.import("model.mdl");
00170      * char* res = new char[11];
00171      * Markov::Random::Mersenne MersenneTwisterEngine;
00172      * for (int i = 0; i < 10; i++) {
00173      *      this->RandomWalk(&MersenneTwisterEngine, 5, 10, res);
00174      *      std::cout « res « "\n";
00175      *  }
00176      * @endcode
00177      *
00178      * @b Example @b Use: Generating a random number with Marsaglia Engine
00179      * @code{.cpp}
00180      * Markov::Random::Mersenne me;
00181      * std::cout « me.random();
00182      * @endcode
00183      *
00184     */
00185     class Mersenne : public DefaultRandomEngine{
00186
00187     };
00188
00189
00190 };
```

9.85 Markopy/MarkovPasswordsGUI/src/about.cpp File Reference

About page.

```
#include <fstream>
#include <QtWidgets/QApplication>
#include "about.h"
```

Include dependency graph for about.cpp:



9.85.1 Detailed Description

About page.

Authors

Yunus Emre Yılmaz

Definition in file about.cpp.

9.86 about.cpp

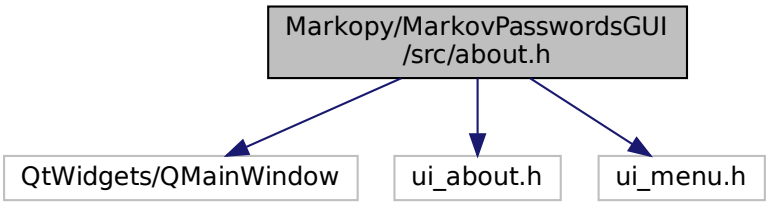
```
00001
00002 /** @file about.cpp
00003  * @brief About page
00004  * @authors Yunus Emre Yılmaz
00005  *
00006  */
00007
00008 #include <fstream>
00009 #include <QtWidgets/QApplication>
00010 #include "about.h"
00011
00012 using namespace Markov::GUI;
00013
00014 about::about(QWidget* parent)
00015     : QMainWindow(parent)
00016 {
00017     ui.setupUi(this);
00018
00019 }
```

9.87 Markopy/MarkovPasswordsGUI/src/about.h File Reference

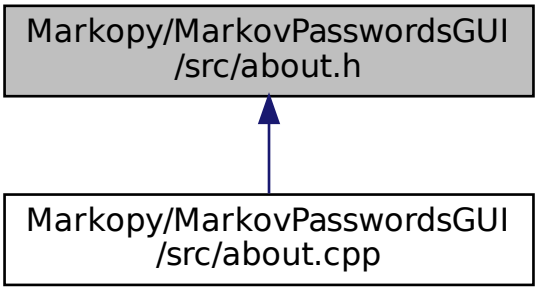
About page.

```
#include <QtWidgets/QMainWindow>
#include "ui_about.h"
#include <ui_menu.h>
```

Include dependency graph for about.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::GUI::about

  QT Class for about page.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::GUI

  namespace for MarkovPasswords API GUI wrapper

9.87.1 Detailed Description

About page.

Authors

Yunus Emre Yılmaz

Definition in file about.h.

9.88 about.h

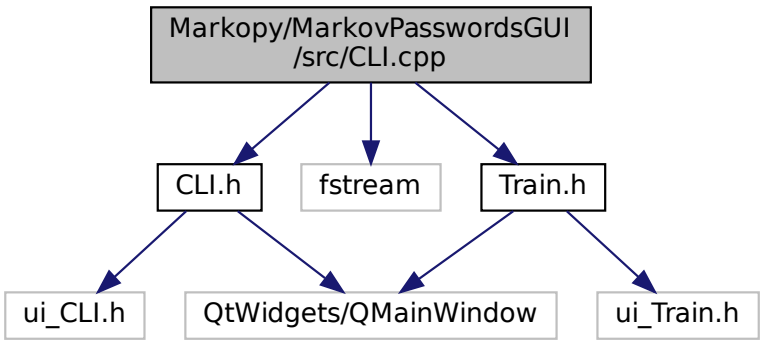
```
00001 /** @file about.h
00002  * @brief About page
00003  * @authors Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 #pragma once
00008 #include <QtWidgets/QMainWindow>
00009 #include "ui_about.h"
00010 #include <ui_menu.h>
00011
00012 /** @brief namespace for MarkovPasswords API GUI wrapper
00013 */
00014 namespace Markov::GUI{
00015
00016     /** @brief QT Class for about page
00017     */
00018     class about :public QMainWindow {
00019     Q_OBJECT
00020     public:
00021         about(QWidget* parent = Q_NULLPTR);
00022
00023     private:
00024         Ui:: main ui;
00025
00026
00027     };
00028 };
```

9.89 Markopy/MarkovPasswordsGUI/src/CLI.cpp File Reference

CLI page.

```
#include "CLI.h"
#include <fstream>
#include "Train.h"
```

Include dependency graph for CLI.cpp:



9.89.1 Detailed Description

CLI page.

Authors

Yunus Emre Yılmaz

Definition in file CLI.cpp.

9.90 CLI.cpp

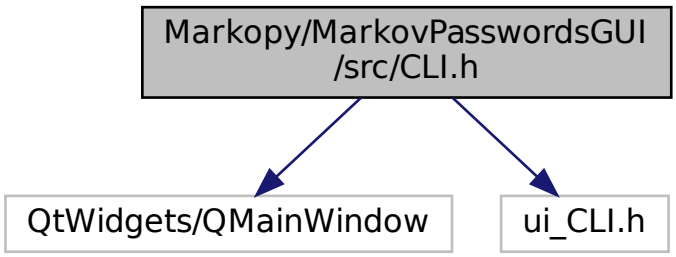
```
00001 /** @file cli.cpp
00002  * @brief CLI page
00003  * @authors Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 #include "CLI.h"
00008 #include <fstream>
00009 #include "Train.h"
00010
00011
00012
00013 using namespace Markov::GUI;
00014
00015 Markov::GUI::CLI::CLI(QWidget* parent)
00016     : QMainWindow(parent)
00017 {
00018     ui.setupUi(this);
00019
00020     QObject::connect(ui.startButton, &QPushButton::clicked, this, [this] {start(); });
00021     QObject::connect(ui.commandLinkButton_2, &QPushButton::clicked, this, [this] {statistics(); });
00022     QObject::connect(ui.commandLinkButton, &QPushButton::clicked, this, [this] {about(); });
00023
00024 }
00025
00026 void Markov::GUI::CLI::start() {
00027     Train* w = new Train;
00028     w->show();
00029     this->close();
00030 }
00031 void Markov::GUI::CLI::statistics() {
00032     /*
00033     statistic will show
00034     */
00035 }
00036 void Markov::GUI::CLI::about() {
00037     /*
00038     about button
00039     */
00040 }
```

9.91 Markopy/MarkovPasswordsGUI/src/CLI.h File Reference

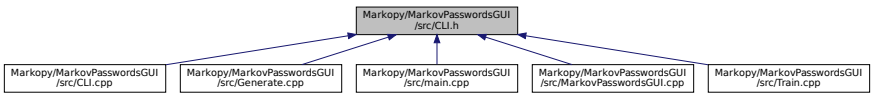
CLI page.

```
#include <QtWidgets/QMainWindow>
#include "ui_CLI.h"
```

Include dependency graph for CLI.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::GUI::CLI

  QT CLI Class.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::GUI

  namespace for MarkovPasswords API GUI wrapper

9.91.1 Detailed Description

CLI page.

Authors

Yunus Emre Yılmaz

Definition in file CLI.h.

9.92 CLI.h

```
00001 /** @file cli.h
00002  * @brief CLI page
00003  * @authors Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 #pragma once
00008 #include <QtWidgets/QMainWindow>
00009 #include "ui_CLI.h"
00010
00011 namespace Markov::GUI{
00012     /** @brief QT CLI Class
00013     */
00014     class CLI :public QMainWindow {
00015         Q_OBJECT
00016     public:
00017         CLI(QWidget* parent = Q_NULLPTR);
00018
00019     private:
00020         Ui::CLI ui;
00021
00022     public slots:
00023         void start();
00024         void statistics();
00025         void about();
00026     };
00027 };
```

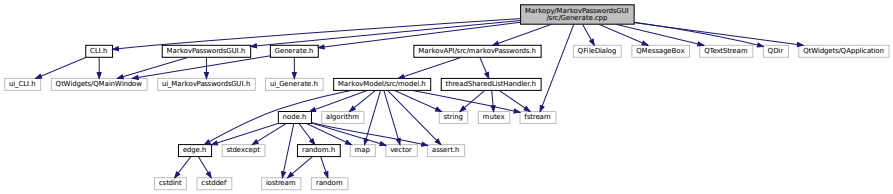
9.93 Markopy/MarkovPasswordsGUI/src/Generate.cpp File Reference

Generation Page.

```
#include "Generate.h"
#include <fstream>
```

```
#include <QFileDialog>
#include <QMessageBox>
#include <QTextStream>
#include <QDir>
#include "CLI.h"
#include "MarkovAPI/src/markovPasswords.h"
#include <QtWidgets/QApplication>
#include "MarkovPasswordsGUI.h"
```

Include dependency graph for Generate.cpp:



9.93.1 Detailed Description

Generation Page.

Authors

Yunus Emre Yılmaz

Definition in file Generate.cpp.

9.94 Generate.cpp

```
00001 /** @file Generate.cpp
00002  * @brief Generation Page
00003  * @authors Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 #include "Generate.h"
00008 #include <fstream>
00009 #include<QFileDialog>
00010 #include<QMessageBox>
00011 #include<QTextStream>
00012 #include<QDir>
00013 #include "CLI.h"
00014 #include "MarkovAPI/src/markovPasswords.h"
00015 #include <QtWidgets/QApplication>
00016 #include "MarkovPasswordsGUI.h"
00017 using namespace Markov::GUI;
00018
00019
00020 Generate::Generate(QWidget* parent)
00021     : QMainWindow(parent)
00022 {
00023     ui.setupUi(this);
00024
00025     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {generation(); });
00026     QObject::connect(ui.pushButton_4, &QPushButton::clicked, this, [this] {home(); });
00027     QObject::connect(ui.pushButton_3, &QPushButton::clicked, this, [this] {train(); });
00028     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {vis(); });
00029
00030
00031     ui.pushButton->setVisible(false);
00032     ui.lineEdit->setVisible(false);
00033     ui.lineEdit_2->setVisible(false);
00034     ui.lineEdit_3->setVisible(false);
00035     ui.label_3->setVisible(false);
00036     ui.label_4->setVisible(false);
00037     ui.label_5->setVisible(false);
00038
00039
00040 }
00041
```

```
00042 void Generate::generation() {
00043
00044
00045
00046
00047     QString file_name = QFileDialog::getOpenFileName(this, "Open a File", QDir::homePath());
00048     QFile file(file_name);
00049
00050
00051
00052     int numberPass = ui.lineEdit->text().toInt();
00053     int minLen = ui.lineEdit_2->text().toInt();
00054     int maxLen = ui.lineEdit_3->text().toInt();
00055     char* cstr;
00056     std::string fname = file_name.toStdString();
00057     cstr = new char[fname.size() + 1];
00058     strcpy(cstr, fname.c_str());
00059
00060     ui.label_6->setText("GENERATING!");
00061
00062     Markov::API::MarkovPasswords mp;
00063     mp.Import("src\\CLI\\sample_models\\2gram-trained.mdl");
00064
00065     mp.Generate(numberPass,cstr,minLen,maxLen);
00066
00067     if (!file.open(QFile::ReadOnly | QFile::Text)) {
00068         QMessageBox::warning(this, "Error", "File Not Open!");
00069     }
00070     QTextStream in(&file);
00071     QString text = in.readAll();
00072     ui.plainTextEdit->setPlainText(text);
00073
00074     ui.label_6->setText("DONE!");
00075
00076
00077
00078     file.close();
00079 }
00080
00081
00082 void Generate::train() {
00083     QString file_name = QFileDialog::getOpenFileName(this, "Open a File", QDir::homePath());
00084     QFile file(file_name);
00085
00086     if (!file.open(QFile::ReadOnly | QFile::Text)) {
00087         QMessageBox::warning(this, "Error", "File Not Open!");
00088     }
00089     QTextStream in(&file);
00090     QString text = in.readAll();
00091
00092
00093     char* cstr;
00094     std::string fname = file_name.toStdString();
00095     cstr = new char[fname.size() + 1];
00096     strcpy(cstr, fname.c_str());
00097
00098
00099
00100     char a = ',';
00101     Markov::API::MarkovPasswords mp;
00102     mp.Import("models\\2gram.mdl");
00103     mp.Train(cstr, a,10);
00104     mp.Export("models\\finished.mdl");
00105
00106
00107
00108     ui.pushButton->setVisible(true);
00109     ui.lineEdit->setVisible(true);
00110     ui.lineEdit_2->setVisible(true);
00111     ui.lineEdit_3->setVisible(true);
00112     ui.label_3->setVisible(true);
00113     ui.label_4->setVisible(true);
00114     ui.label_5->setVisible(true);
00115
00116     file.close();
00117
00118
00119 }
00120
00121 void Generate::home() {
00122      CLI* w = new CLI;
00123      w->show();
00124      this->close();
00125 }
00126 void Generate :: vis() {
00127     MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
00128     w->show();
```

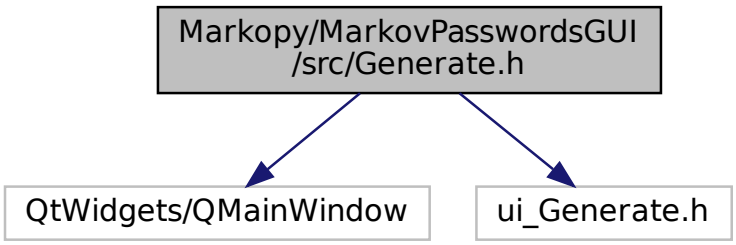
```
00129     this->close();
00130 }
```

9.95 Markopy/MarkovPasswordsGUI/src/Generate.h File Reference

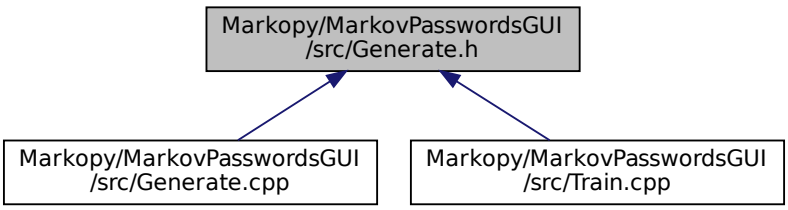
Generation Page.

```
#include <QtWidgets/QMainWindow>
#include "ui_Generate.h"
```

Include dependency graph for Generate.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::GUI::Generate

  QT Generation page class.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::GUI

  namespace for MarkovPasswords API GUI wrapper

9.95.1 Detailed Description

Generation Page.

Authors

Yunus Emre Yılmaz

Definition in file Generate.h.

9.96 Generate.h

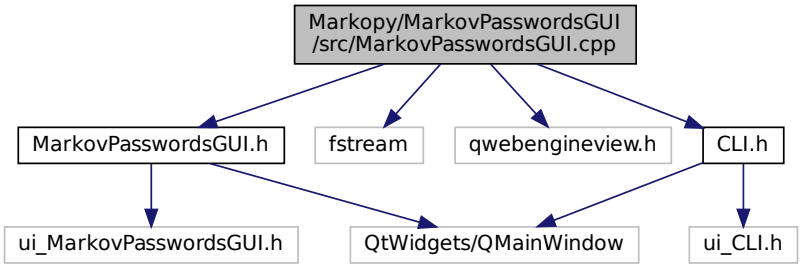
```
00001 /** @file Generate.h
00002  * @brief Generation Page
00003  * @authors Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 #pragma once
00008 #include <QtWidgets/QMainWindow>
00009 #include "ui_Generate.h"
00010
00011
00012 namespace Markov::GUI{
00013     /** @brief QT Generation page class
00014     */
00015     class Generate :public QMainWindow {
00016         Q_OBJECT
00017     public:
00018         Generate(QWidget* parent = Q_NULLPTR);
00019
00020     private:
00021         Ui::Generate ui;
00022
00023     public slots:
00024     void home();
00025     void generation();
00026     void train();
00027     void vis();
00028     };
00029 };
```

9.97 Markopy/MarkovPasswordsGUI/src/MarkovPasswordsGUI.cpp File Reference

Main activity page for GUI.

```
#include "MarkovPasswordsGUI.h"
#include <fstream>
#include <qwebengineview.h>
#include "CLI.h"
```

Include dependency graph for MarkovPasswordsGUI.cpp:



9.97.1 Detailed Description

Main activity page for GUI.

Authors

Yunus Emre Yılmaz

Definition in file MarkovPasswordsGUI.cpp.

9.98 MarkovPasswordsGUI.cpp

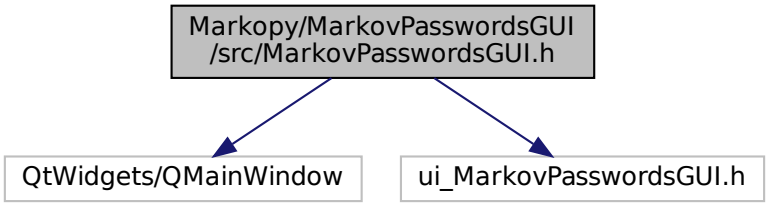
```
00001 /** @file MarkovPasswordsGUI.cpp
00002  * @brief Main activity page for GUI
00003  * @authors Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 #include "MarkovPasswordsGUI.h"
00008 #include <fstream>
00009 #include <qwebengineview.h>
00010 #include "CLI.h"
00011
00012 using namespace Markov::GUI;
00013
00014 Markov::GUI::MarkovPasswordsGUI::MarkovPasswordsGUI(QWidget *parent) : QMainWindow(parent){
00015     ui.setupUi(this);
00016
00017
00018     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {home(); });
00019     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {model(); });
00020     QObject::connect(ui.pushButton_3, &QPushButton::clicked, this, [this] {pass(); });
00021 }
00022
00023
00024 void MarkovPasswordsGUI::home() {
00025     CLI* w = new CLI;
00026     w->show();
00027     this->close();
00028 }
00029 void MarkovPasswordsGUI::pass() {
00030     QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00031
00032     //get working directory
00033     char path[255];
00034     GetCurrentDirectoryA(255, path);
00035
00036     //get absolute path to the layout html
00037     std::string layout = "file:///" + std::string(path) + "\\views\\bar.html";
00038     std::replace(layout.begin(), layout.end(), '\\', '/');
00039     webkit->setUrl(QUrl(layout.c_str()));
00040 }
00041
00042 void MarkovPasswordsGUI::model() {
00043     QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00044
00045     //get working directory
00046     char path[255];
00047     GetCurrentDirectoryA(255, path);
00048
00049     //get absolute path to the layout html
00050     std::string layout = "file:///" + std::string(path) + "\\views\\index.html";
00051     std::replace(layout.begin(), layout.end(), '\\', '/');
00052     webkit->setUrl(QUrl(layout.c_str()));
00053 }
```

9.99 Markopy/MarkovPasswordsGUI/src/MarkovPasswordsGUI.h File Reference

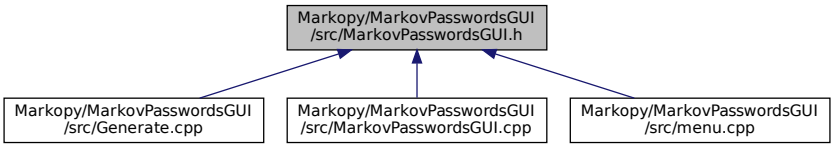
Main activity page for GUI.

```
#include <QtWidgets/QMainWindow>
#include "ui_MarkovPasswordsGUI.h"
```

Include dependency graph for MarkovPasswordsGUI.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::GUI::MarkovPasswordsGUI

  Reporting UI.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::GUI

  namespace for MarkovPasswords API GUI wrapper

9.99.1 Detailed Description

Main activity page for GUI.

Authors

Yunus Emre Yılmaz

Definition in file MarkovPasswordsGUI.h.

9.100 MarkovPasswordsGUI.h

```
00001 /** @file MarkovPasswordsGUI.h
00002  * @brief Main activity page for GUI
00003  * @authors Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 #pragma once
00008
00009 #include <QtWidgets/QMainWindow>
00010 #include "ui_MarkovPasswordsGUI.h"
```

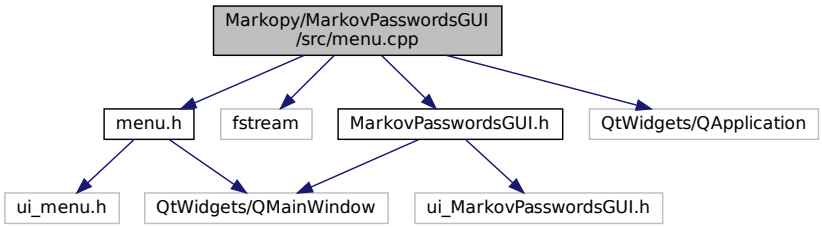
```
00011
00012
00013
00014 namespace Markov::GUI{
00015     /** @brief Reporting UI.
00016     *
00017     * UI for reporting and debugging tools for MarkovPassword
00018     */
00019     class MarkovPasswordsGUI : public QMainWindow {
00020         Q_OBJECT
00021     public:
00022         MarkovPasswordsGUI(QWidget* parent = Q_NULLPTR);
00023
00024     private:
00025         Ui::MarkovPasswordsGUIClass ui;
00026
00027
00028         //Slots for buttons in GUI.
00029     public slots:
00030
00031         void benchmarkSelected();
00032         //void MarkovPasswordsGUI::modelvisSelected();
00033         //void MarkovPasswordsGUI::visualDebugSelected();
00034         //void MarkovPasswordsGUI::comparisonSelected();
00035
00036
00037   public slots:
00038
00039       void home();
00040       void pass();
00041       void model();
00042   };
00043 };
```

9.101 Markopy/MarkovPasswordsGUI/src/menu.cpp File Reference

menu page

```
#include "menu.h"
#include <fstream>
#include "MarkovPasswordsGUI.h"
#include <QtWidgets/QApplication>
```

Include dependency graph for menu.cpp:



9.101.1 Detailed Description

menu page

Authors

Yunus Emre Yılmaz

Definition in file menu.cpp.

9.102 menu.cpp

```
00001 /** @file menu.cpp
00002  * @brief menu page
```

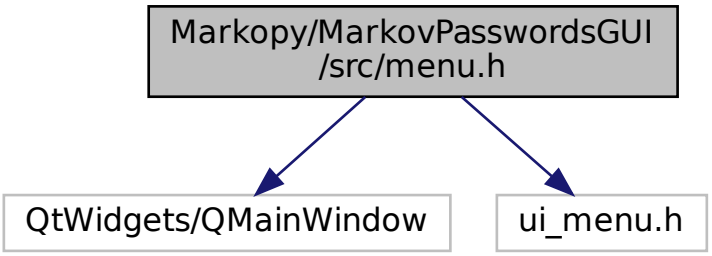
```
00003  * @authors Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 #include "menu.h"
00008 #include <fstream>
00009 #include "MarkovPasswordsGUI.h"
00010 #include <QtWidgets/QApplication>
00011
00012 using namespace Markov::GUI;
00013
00014 menu::menu(QWidget* parent)
00015     : QMainWindow(parent)
00016 {
00017     ui.setupUi(this);
00018
00019
00020     //QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {about(); });
00021     QObject::connect(ui.visu, &QPushButton::clicked, this, [this] {visualization(); });
00022 }
00023 void menu::about() {
00024
00025
00026 }
00027 void menu::visualization() {
00028     MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
00029     w->show();
00030     this->close();
00031 }
```

9.103 Markopy/MarkovPasswordsGUI/src/menu.h File Reference

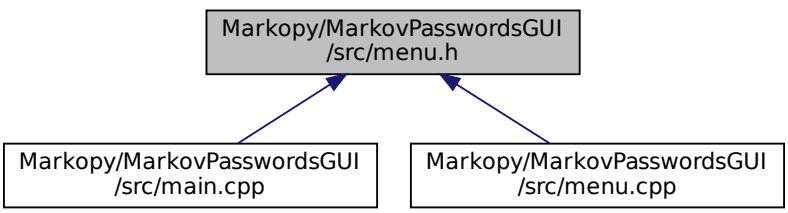
menu page

```
#include <QtWidgets/QMainWindow>
#include "ui_menu.h"
```

Include dependency graph for menu.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::GUI::menu

  QT Menu class.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::GUI

  namespace for MarkovPasswords API GUI wrapper

9.103.1 Detailed Description

menu page

Authors

Yunus Emre Yılmaz

Definition in file menu.h.

9.104 menu.h

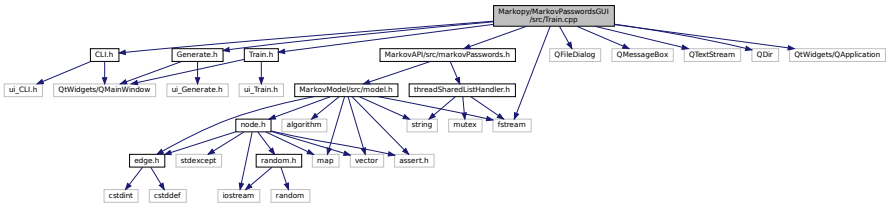
```
00001 /** @file menu.h
00002  * @brief menu page
00003  * @authors Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 #pragma once
00008 #include <QtWidgets/QMainWindow>
00009 #include "ui_menu.h"
00010
00011
00012 namespace Markov::GUI{
00013     /** @brief QT Menu class
00014     */
00015     class menu:public QMainWindow {
00016     Q_OBJECT
00017     public:
00018         menu(QWidget* parent = Q_NULLPTR);
00019
00020     private:
00021         Ui::main ui;
00022
00023     public slots:
00024         void about();
00025         void visualization();
00026     };
00027 };
```

9.105 Markopy/MarkovPasswordsGUI/src/Train.cpp File Reference

training page for GUI

```
#include "Train.h"
#include <fstream>
#include <QFileDialog>
#include <QMessageBox>
#include <QTextStream>
#include <QDir>
#include "CLI.h"
#include "MarkovAPI/src/markovPasswords.h"
#include <QtWidgets/QApplication>
```

`#include "Generate.h"`
Include dependency graph for Train.cpp:



9.105.1 Detailed Description

training page for GUI

Authors

Yunus Emre Yılmaz

Definition in file Train.cpp.

9.106 Train.cpp

```
00001 /** @file Train.cpp
00002  * @brief training page for GUI
00003  * @authors Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 #include "Train.h"
00008 #include <fstream>
00009 #include<QFileDialog>
00010 #include<QMessageBox>
00011 #include<QTextStream>
00012 #include<QDir>
00013 #include "CLI.h"
00014 #include "MarkovAPI/src/markovPasswords.h"
00015
00016 #include <QtWidgets/QApplication>
00017 #include "Generate.h"
00018
00019
00020 using namespace Markov::GUI;
00021
00022 Markov::GUI::Train::Train(QWidget* parent)
00023     : QMainWindow(parent)
00024 {
00025     ui.setupUi(this);
00026
00027
00028
00029     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {train(); });
00030     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {home(); });
00031     //QObject::connect(ui.pushButton_3, &QPushButton::clicked, this, [this] {goGenerate(); });
00032
00033     //ui.pushButton_3->setVisible(false);
00034
00035
00036 }
00037
00038 void Markov::GUI::Train::train() {
00039
00040
00041
00042     QString file_name = QFileDialog::getOpenFileName(this, "Open a File", QDir::homePath());
00043     QFile file(file_name);
00044
00045     if (!file.open(QFile::ReadOnly | QFile::Text)) {
00046         QMessageBox::warning(this, "Error", "File Not Open!");
00047     }
00048     QTextStream in(&file);
00049     QString text = in.readAll();
00050     ui.plainTextEdit->setPlainText(text);
00051
```

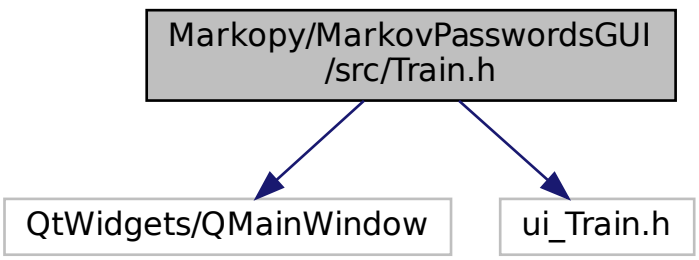
```
00052
00053     char* cstr;
00054     std::string fname = file_name.toStdString();
00055     cstr = new char[fname.size() + 1];
00056     strcpy(cstr, fname.c_str());
00057
00058
00059
00060     char a=',';
00061     Markov::API::MarkovPasswords mp;
00062     mp.Import("models/2gram.mdl");
00063     mp.Train(cstr, a, 10); //please parameterize this hardcoded 10 threads
00064     mp.Export("models/finished.mdl");
00065
00066     ui.label_2->setText("Training DONE!");
00067     //ui.pushButton_3->setVisible(true);
00068
00069
00070     file.close();
00071 }
00072
00073 void Markov::GUI::Train::home() {
00074      CLI* w = new CLI;
00075      w->show();
00076      this->close();
00077 }
00078 /*void Train::goGenerate() {
00079     Generate* w = new Generate;
00080     w->show();
00081     this->close();
00082 }*/
```

9.107 Markopy/MarkovPasswordsGUI/src/Train.h File Reference

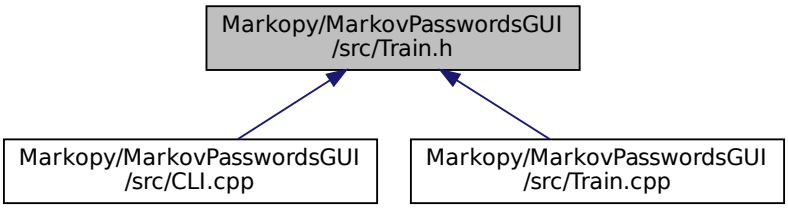
training page for GUI

```
#include <QtWidgets/QMainWindow>
#include "ui_Train.h"
```

Include dependency graph for Train.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Markov::GUI::Train

  QT Training page class.

Namespaces

- Markov

  Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- Markov::GUI

  namespace for MarkovPasswords API GUI wrapper

9.107.1 Detailed Description

training page for GUI

Authors

Yunus Emre Yılmaz

Definition in file Train.h.

9.108 Train.h

```
00001 /** @file Train.h
00002  * @brief training page for GUI
00003  * @authors Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 #pragma once
00008 #include <QtWidgets/QMainWindow>
00009 #include "ui_Train.h"
00010
00011 namespace Markov::GUI{
00012
00013     /** @brief QT Training page class
00014     */
00015     class Train :public QMainWindow {
00016     Q_OBJECT
00017     public:
00018         Train(QWidget* parent = Q_NULLPTR);
00019
00020     private:
00021         Ui::Train ui;
00022
00023     public slots:
00024         void home();
00025         void train();
00026     };
00027 };
```

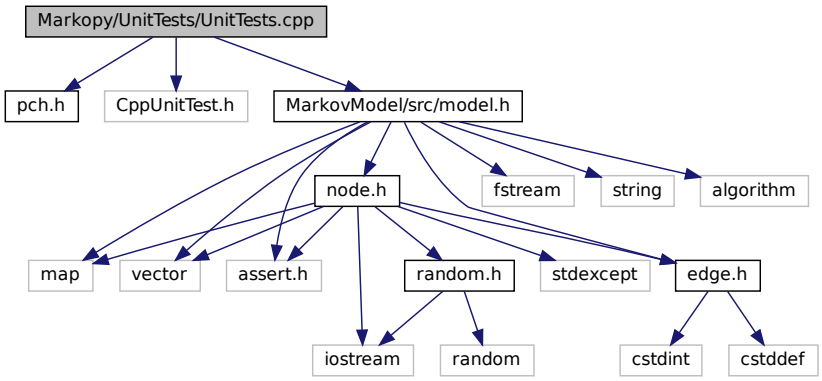
9.109 Markopy/README.md File Reference

9.110 Markopy/UnitTests/UnitTests.cpp File Reference

Unit tests with Microsoft::VisualStudio::CppUnitTestFramework.

```
#include "pch.h"
#include "CppUnitTest.h"
#include "MarkovModel/src/model.h"
```

Include dependency graph for UnitTests.cpp:



Namespaces

- Testing

  Namespace for Microsoft Native Unit Testing Classes.
- Testing::MVP

  Testing Namespace for Minimal Viable Product.
- Testing::MVP::MarkovModel

  Testing Namespace for MVP MarkovModel.
- Testing::MVP::MarkovPasswords

  Testing namespace for MVP MarkovPasswords.
- Testing::MarkovModel

  Testing namespace for MarkovModel.
- Testing::MarkovPasswords

  Testing namespace for MarkovPasswords.

Functions

- Testing::MVP::MarkovModel::TEST_CLASS (Edge)

  Test class for minimal viable Edge.
- Testing::MVP::MarkovModel::TEST_CLASS (Node)

  Test class for minimal viable Node.
- Testing::MVP::MarkovModel::TEST_CLASS (Model)

  Test class for minimal viable Model.
- Testing::MVP::MarkovPasswords::TEST_CLASS (ArgParser)

  Test Class for Argparse class.
- Testing::MarkovModel::TEST_CLASS (Edge)

  Test class for rest of Edge cases.

- Testing::MarkovModel::TEST_CLASS (Node)

  Test class for rest of Node cases.
- Testing::MarkovModel::TEST_CLASS (Model)

  Test class for rest of model cases.

9.110.1 Detailed Description

Unit tests with Microsoft::VisualStudio::CppUnitTestFramework.

Authors

Ata Hakçıl, Osman Ömer Yıldıztugay, Yunus Emre Yılmaz

Definition in file UnitTests.cpp.

9.111 UnitTests.cpp

```
00001 /** @file UnitTests.cpp
00002  * @brief Unit tests with Microsoft::VisualStudio::CppUnitTestFramework
00003  * @authors Ata Hakçıl, Osman Ömer Yıldıztugay, Yunus Emre Yılmaz
00004  *
00005  */
00006
00007 #include "pch.h"
00008 #include "CppUnitTest.h"
00009 #include "MarkovModel/src/model.h"
00010
00011 using namespace Microsoft::VisualStudio::CppUnitTestFramework;
00012
00013
00014 /** @brief Namespace for Microsoft Native Unit Testing Classes
00015 */
00016 namespace Testing {
00017
00018     /** @brief Testing Namespace for Minimal Viable Product
00019     */
00020     namespace MVP {
00021         /** @brief Testing Namespace for MVP MarkovModel
00022         */
00023         namespace MarkovModel
00024         {
00025             /** @brief Test class for minimal viable Edge
00026             */
00027             TEST_CLASS(Edge)
00028             {
00029             public:
00030
00031                 /** @brief test default constructor
00032                 */
00033                 TEST_METHOD(default_constructor) {
00034                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>;
00035                     Assert::IsNull(e->LeftNode());
00036                     Assert::IsNull(e->RightNode());
00037                     delete e;
00038                 }
00039
00040                 /** @brief test linked constructor with two nodes
00041                 */
00042                 TEST_METHOD(linked_constructor) {
00043                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00044                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00045                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
      RightNode);
00046                     Assert::IsTrue(LeftNode == e->LeftNode());
00047                     Assert::IsTrue(RightNode == e->RightNode());
00048                     delete LeftNode;
00049                     delete RightNode;
00050                     delete e;
00051                 }
00052
00053                 /** @brief test AdjustEdge function
00054                 */
00055                 TEST_METHOD(AdjustEdge) {
00056                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00057                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00058                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
      RightNode);
00059                     e->AdjustEdge(15);
00060                     Assert::AreEqual(15ull, e->EdgeWeight());
```

```
00061                   e->AdjustEdge(15);
00062                   Assert::AreEqual(30ull, e->EdgeWeight());
00063                   delete LeftNode;
00064                   delete RightNode;
00065                   delete e;
00066               }
00067
00068               /** @brief test TraverseNode returning RightNode
00069               */
00070               TEST_METHOD(TraverseNode) {
00071                   Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00072                   Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00073                   Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
    RightNode);
00074                   Assert::IsTrue(RightNode == e->TraverseNode());
00075                   delete LeftNode;
00076                   delete RightNode;
00077                   delete e;
00078               }
00079
00080               /** @brief test LeftNode/RightNode setter
00081               */
00082               TEST_METHOD(set_left_and_right) {
00083                   Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00084                   Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00085                   Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(LeftNode,
    RightNode);
00086
00087                   Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>;
00088                   e2->SetLeftEdge(LeftNode);
00089                   e2->SetRightEdge(RightNode);
00090
00091                   Assert::IsTrue(e1->LeftNode() == e2->LeftNode());
00092                   Assert::IsTrue(e1->RightNode() == e2->RightNode());
00093                   delete LeftNode;
00094                   delete RightNode;
00095                   delete e1;
00096                   delete e2;
00097               }
00098
00099               /** @brief test negative adjustments
00100               */
00101               TEST_METHOD(negative_adjust) {
00102                   Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00103                   Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00104                   Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
    RightNode);
00105                   e->AdjustEdge(15);
00106                   Assert::AreEqual(15ull, e->EdgeWeight());
00107                   e->AdjustEdge(-15);
00108                   Assert::AreEqual(0ull, e->EdgeWeight());
00109                   delete LeftNode;
00110                   delete RightNode;
00111                   delete e;
00112               }
00113           };
00114
00115           /** @brief Test class for minimal viable Node
00116           */
00117           TEST_CLASS(Node)
00118           {
00119           public:
00120
00121               /** @brief test default constructor
00122               */
00123               TEST_METHOD(default_constructor) {
00124                   Markov::Node<unsigned char>* n = new Markov::Node<unsigned char>();
00125                   Assert::AreEqual((unsigned char)0, n->NodeValue());
00126                   delete n;
00127               }
00128
00129               /** @brief test custom constructor with unsigned char
00130               */
00131               TEST_METHOD(uchar_constructor) {
00132                   Markov::Node<unsigned char>* n = NULL;
00133                   unsigned char test_cases[] = { 'c', 0x00, 0xff, -32 };
00134                   for (unsigned char tcase : test_cases) {
00135                       n = new Markov::Node<unsigned char>(tcase);
00136                       Assert::AreEqual(tcase, n->NodeValue());
00137                       delete n;
00138                   }
00139               }
00140
00141               /** @brief test link function
00142               */
00143               TEST_METHOD(link_left) {
00144                   Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
```

```
00145                   Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00146
00147                   Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
00148                   delete LeftNode;
00149                   delete RightNode;
00150                   delete e;
00151               }
00152
00153               /** @brief test link function
00154               */
00155               TEST_METHOD(link_right) {
00156                   Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00157                   Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00158
00159                   Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(NULL, RightNode);
00160                   LeftNode->Link(e);
00161                   Assert::IsTrue(LeftNode == e->LeftNode());
00162                   Assert::IsTrue(RightNode == e->RightNode());
00163                   delete LeftNode;
00164                   delete RightNode;
00165                   delete e;
00166               }
00167
00168               /** @brief test RandomNext with low values
00169               */
00170               TEST_METHOD(rand_next_low) {
00171                   Markov::Random::Marsaglia MarsagliaRandomEngine;
00172                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00173                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00174                   Markov::Edge<unsigned char>* e = src->Link(target1);
00175                   e->AdjustEdge(15);
00176                   Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00177                   Assert::IsTrue(res == target1);
00178                   delete src;
00179                   delete target1;
00180                   delete e;
00181
00182               }
00183
00184               /** @brief test RandomNext with 32 bit high values
00185               */
00186               TEST_METHOD(rand_next_u32) {
00187                   Markov::Random::Marsaglia MarsagliaRandomEngine;
00188                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00189                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00190                   Markov::Edge<unsigned char>* e = src->Link(target1);
00191                   e->AdjustEdge(1 « 31);
00192                   Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00193                   Assert::IsTrue(res == target1);
00194                   delete src;
00195                   delete target1;
00196                   delete e;
00197
00198               }
00199
00200               /** @brief random next on a node with no follow-ups
00201               */
00202               TEST_METHOD(rand_next_choice_1) {
00203                   Markov::Random::Marsaglia MarsagliaRandomEngine;
00204                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00205                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00206                   Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00207                   Markov::Edge<unsigned char>* e1 = src->Link(target1);
00208                   Markov::Edge<unsigned char>* e2 = src->Link(target2);
00209                   e1->AdjustEdge(1);
00210                   e2->AdjustEdge((unsigned long)(1ull « 31));
00211                   Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00212                   Assert::IsNotNull(res);
00213                   Assert::IsTrue(res == target2);
00214                   delete src;
00215                   delete target1;
00216                   delete e1;
00217                   delete e2;
00218               }
00219
00220               /** @brief random next on a node with no follow-ups
00221               */
00222               TEST_METHOD(rand_next_choice_2) {
00223                   Markov::Random::Marsaglia MarsagliaRandomEngine;
00224                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00225                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00226                   Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00227                   Markov::Edge<unsigned char>* e1 = src->Link(target1);
00228                   Markov::Edge<unsigned char>* e2 = src->Link(target2);
00229                   e2->AdjustEdge(1);
00230                   e1->AdjustEdge((unsigned long)(1ull « 31));
00231                   Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
```

```
00232                   Assert::IsNotNull(res);
00233                   Assert::IsTrue(res == target1);
00234                   delete src;
00235                   delete target1;
00236                   delete e1;
00237                   delete e2;
00238               }
00239
00240
00241               /** @brief test updateEdges
00242               */
00243               TEST_METHOD(update_edges_count) {
00244
00245                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00246                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00247                   Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00248                   Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00249                   Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00250                   e1->AdjustEdge(25);
00251                   src->UpdateEdges(e1);
00252                   e2->AdjustEdge(30);
00253                   src->UpdateEdges(e2);
00254
00255                   Assert::AreEqual((size_t)2, src->Edges()->size());
00256
00257                   delete src;
00258                   delete target1;
00259                   delete e1;
00260                   delete e2;
00261
00262               }
00263
00264               /** @brief test updateEdges
00265               */
00266               TEST_METHOD(update_edges_total) {
00267
00268                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00269                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00270                   Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00271                   Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target1);
00272                   e1->AdjustEdge(25);
00273                   src->UpdateEdges(e1);
00274                   e2->AdjustEdge(30);
00275                   src->UpdateEdges(e2);
00276
00277                   //Assert::AreEqual(55ull, src->TotalEdgeWeights());
00278
00279                   delete src;
00280                   delete target1;
00281                   delete e1;
00282                   delete e2;
00283               }
00284
00285
00286               /** @brief test FindVertice
00287               */
00288               TEST_METHOD(find_vertice) {
00289
00290                   Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00291                   Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00292                   Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00293                   Markov::Edge<unsigned char>* res = NULL;
00294                   src->Link(target1);
00295                   src->Link(target2);
00296
00297
00298                   res = src->FindEdge('b');
00299                   Assert::IsNotNull(res);
00300                   Assert::AreEqual((unsigned char)'b', res->TraverseNode()->NodeValue());
00301                   res = src->FindEdge('c');
00302                   Assert::IsNotNull(res);
00303                   Assert::AreEqual((unsigned char)'c', res->TraverseNode()->NodeValue());
00304
00305                   delete src;
00306                   delete target1;
00307                   delete target2;
00308
00309
00310               }
00311
00312
00313               /** @brief test FindVertice
00314               */
00315               TEST_METHOD(find_vertice_without_any) {
00316
00317                   auto _invalid_next = [] {
00318                       Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
```

```
00319                         Markov::Edge<unsigned char>* res = NULL;
00320
00321                         res = src->FindEdge('b');
00322                         Assert::IsNull(res);
00323
00324                         delete src;
00325                     };
00326
00327                     //Assert::ExpectException<std::logic_error>(_invalid_next);
00328                 }
00329
00330                 /** @brief test FindVertice
00331                 */
00332                 TEST_METHOD(find_vertice_nonexistent) {
00333
00334                     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00335                     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00336                     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00337                     Markov::Edge<unsigned char>* res = NULL;
00338                     src->Link(target1);
00339                     src->Link(target2);
00340
00341                     res = src->FindEdge('D');
00342                     Assert::IsNull(res);
00343
00344                     delete src;
00345                     delete target1;
00346                     delete target2;
00347
00348                 }
00349             };
00350
00351             /** @brief Test class for minimal viable Model
00352             */
00353             TEST_CLASS(Model)
00354             {
00355             public:
00356                 /** @brief test model constructor for starter node
00357                 */
00358                 TEST_METHOD(model_constructor) {
00359                     Markov::Model<unsigned char> m;
00360                     Assert::AreEqual((unsigned char)'\0', m.StarterNode()->NodeValue());
00361                 }
00362
00363                 /** @brief test import
00364                 */
00365                 TEST_METHOD(import_filename) {
00366                     Markov::Model<unsigned char> m;
00367                     Assert::IsTrue(m.Import("../MarkovPasswords/Models/2gram.mdl"));
00368                 }
00369
00370                 /** @brief test export
00371                 */
00372                 TEST_METHOD(export_filename) {
00373                     Markov::Model<unsigned char> m;
00374                     Assert::IsTrue(m.Export("../MarkovPasswords/Models/testcase.mdl"));
00375                 }
00376
00377                 /** @brief test random walk
00378                 */
00379                 TEST_METHOD(random_walk) {
00380                     unsigned char* res = new unsigned char[12 + 5];
00381                     Markov::Random::Marsaglia MarsagliaRandomEngine;
00382                     Markov::Model<unsigned char> m;
00383                     Assert::IsTrue(m.Import("../Models/finished2.mdl"));
00384                     Assert::IsNotNull(m.RandomWalk(&MarsagliaRandomEngine,1,12,res));
00385                 }
00386             };
00387         }
00388
00389         /** @brief Testing namespace for MVP MarkovPasswords
00390         */
00391         namespace MarkovPasswords
00392         {
00393             /** @brief Test Class for Argparse class
00394             */
00395             TEST_CLASS(ArgParser)
00396             {
00397             public:
00398                 /** @brief test basic generate
00399                 */
00400                 TEST_METHOD(generate_basic) {
00401                     int argc = 8;
00402                     char *argv[] = {"markov.exe", "generate", "-if", "model.mdl", "-of",
       "passwords.txt", "-n", "100"};
00403
00404                     /*ProgramOptions *p = Argparse::parse(argc, argv);
```

```
00405                     Assert::IsNotNull(p);
00406
00407                     Assert::AreEqual(p->bImport, true);
00408                     Assert::AreEqual(p->bExport, false);
00409                     Assert::AreEqual(p->importname, "model.mdl");
00410                     Assert::AreEqual(p->outputfilename, "passwords.txt");
00411                     Assert::AreEqual(p->generateN, 100); */
00412
00413                 }
00414
00415                 /** @brief test basic generate reordered params
00416                 */
00417                 TEST_METHOD(generate_basic_reorder) {
00418                     int argc = 8;
00419                     char *argv[] = { "markov.exe", "generate", "-n", "100", "-if", "model.mdl", "-of",
      "passwords.txt" };
00420
00421                     /*ProgramOptions* p = Argparse::parse(argc, argv);
00422                     Assert::IsNotNull(p);
00423
00424                     Assert::AreEqual(p->bImport, true);
00425                     Assert::AreEqual(p->bExport, false);
00426                     Assert::AreEqual(p->importname, "model.mdl");
00427                     Assert::AreEqual(p->outputfilename, "passwords.txt");
00428                     Assert::AreEqual(p->generateN, 100);*/
00429                 }
00430
00431                 /** @brief test basic generate param longnames
00432                 */
00433                 TEST_METHOD(generate_basic_longname) {
00434                     int argc = 8;
00435                     char *argv[] = { "markov.exe", "generate", "-n", "100", "--inputfilename",
      "model.mdl", "--outputfilename", "passwords.txt" };
00436
00437                     /*ProgramOptions* p = Argparse::parse(argc, argv);
00438                     Assert::IsNotNull(p);
00439
00440                     Assert::AreEqual(p->bImport, true);
00441                     Assert::AreEqual(p->bExport, false);
00442                     Assert::AreEqual(p->importname, "model.mdl");
00443                     Assert::AreEqual(p->outputfilename, "passwords.txt");
00444                     Assert::AreEqual(p->generateN, 100); */
00445                 }
00446
00447                 /** @brief test basic generate
00448                 */
00449                 TEST_METHOD(generate_fail_badmethod) {
00450                     int argc = 8;
00451                     char *argv[] = { "markov.exe", "junk", "-n", "100", "--inputfilename",
      "model.mdl", "--outputfilename", "passwords.txt" };
00452
00453                     /*ProgramOptions* p = Argparse::parse(argc, argv);
00454                     Assert::IsNull(p); */
00455                 }
00456
00457                 /** @brief test basic train
00458                 */
00459                 TEST_METHOD(train_basic) {
00460                     int argc = 4;
00461                     char *argv[] = { "markov.exe", "train", "-ef", "model.mdl" };
00462
00463                     /*ProgramOptions* p = Argparse::parse(argc, argv);
00464                     Assert::IsNotNull(p);
00465
00466                     Assert::AreEqual(p->bImport, false);
00467                     Assert::AreEqual(p->bExport, true);
00468                     Assert::AreEqual(p->exportname, "model.mdl"); */
00469
00470                 }
00471
00472                 /** @brief test basic generate
00473                 */
00474                 TEST_METHOD(train_basic_longname) {
00475                     int argc = 4;
00476                     char *argv[] = { "markov.exe", "train", "--exportfilename", "model.mdl" };
00477
00478                     /*ProgramOptions* p = Argparse::parse(argc, argv);
00479                     Assert::IsNotNull(p);
00480
00481                     Assert::AreEqual(p->bImport, false);
00482                     Assert::AreEqual(p->bExport, true);
00483                     Assert::AreEqual(p->exportname, "model.mdl"); */
00484                 }
00485
00486
00487
00488             };
```

```
00489
00490         }
00491     }
00492
00493
00494     /** @brief Testing namespace for MarkovModel
00495     */
00496     namespace MarkovModel {
00497
00498         /** @brief Test class for rest of Edge cases
00499         */
00500         TEST_CLASS(Edge)
00501         {
00502         public:
00503             /** @brief send exception on integer underflow
00504             */
00505             TEST_METHOD(except_integer_underflow) {
00506                 auto _underflow_adjust = [] {
00507                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00508                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00509                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
    RightNode);
00510                     e->AdjustEdge(15);
00511                     e->AdjustEdge(-30);
00512                     delete LeftNode;
00513                     delete RightNode;
00514                     delete e;
00515                 };
00516                 Assert::ExpectException<std::underflow_error>(_underflow_adjust);
00517             }
00518
00519             /** @brief test integer overflows
00520             */
00521             TEST_METHOD(except_integer_overflow) {
00522                 auto _overflow_adjust = [] {
00523                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00524                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00525                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
    RightNode);
00526                     e->AdjustEdge(~0ull);
00527                     e->AdjustEdge(1);
00528                     delete LeftNode;
00529                     delete RightNode;
00530                     delete e;
00531                 };
00532                 Assert::ExpectException<std::underflow_error>(_overflow_adjust);
00533             }
00534         };
00535
00536         /** @brief Test class for rest of Node cases
00537         */
00538         TEST_CLASS(Node)
00539         {
00540         public:
00541
00542             /** @brief test RandomNext with 64 bit high values
00543             */
00544             TEST_METHOD(rand_next_u64) {
00545                 Markov::Random::Marsaglia MarsagliaRandomEngine;
00546                 Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00547                 Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00548                 Markov::Edge<unsigned char>* e = src->Link(target1);
00549                 e->AdjustEdge((unsigned long)(1ull << 63));
00550                 Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00551                 Assert::IsTrue(res == target1);
00552                 delete src;
00553                 delete target1;
00554                 delete e;
00555
00556             }
00557
00558             /** @brief test RandomNext with 64 bit high values
00559             */
00560             TEST_METHOD(rand_next_u64_max) {
00561                 Markov::Random::Marsaglia MarsagliaRandomEngine;
00562                 Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00563                 Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00564                 Markov::Edge<unsigned char>* e = src->Link(target1);
00565                 e->AdjustEdge((0xffffFFFF));
00566                 Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00567                 Assert::IsTrue(res == target1);
00568                 delete src;
00569                 delete target1;
00570                 delete e;
00571
00572             }
00573
```

```
00574             /** @brief randomNext when no edges are present
00575             */
00576             TEST_METHOD(uninitialized_rand_next) {
00577 
00578                 auto _invalid_next = [] {
00579                     Markov::Random::Marsaglia MarsagliaRandomEngine;
00580                     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00581                     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00582                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(src, target1);
00583                     Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00584 
00585                     delete src;
00586                     delete target1;
00587                     delete e;
00588                 };
00589 
00590                 Assert::ExpectException<std::logic_error>(_invalid_next);
00591             }
00592 
00593 
00594         };
00595 
00596         /** @brief Test class for rest of model cases
00597         */
00598         TEST_CLASS(Model)
00599         {
00600         public:
00601             TEST_METHOD(functional_random_walk) {
00602                 unsigned char* res2 = new unsigned char[12 + 5];
00603                 Markov::Random::Marsaglia MarsagliaRandomEngine;
00604                 Markov::Model<unsigned char> m;
00605                 Markov::Node<unsigned char>* starter = m.StarterNode();
00606                 Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00607                 Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
00608                 Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00609                 Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
00610                 starter->Link(a)->AdjustEdge(1);
00611                 a->Link(b)->AdjustEdge(1);
00612                 b->Link(c)->AdjustEdge(1);
00613                 c->Link(end)->AdjustEdge(1);
00614 
00615                 char* res = (char*)m.RandomWalk(&MarsagliaRandomEngine,1,12,res2);
00616                 Assert::IsFalse(strcmp(res, "abc"));
00617             }
00618             TEST_METHOD(functionoal_random_walk_without_any) {
00619                 Markov::Model<unsigned char> m;
00620                 Markov::Node<unsigned char>* starter = m.StarterNode();
00621                 Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00622                 Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
00623                 Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00624                 Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
00625                 Markov::Edge<unsigned char>* res = NULL;
00626                 starter->Link(a)->AdjustEdge(1);
00627                 a->Link(b)->AdjustEdge(1);
00628                 b->Link(c)->AdjustEdge(1);
00629                 c->Link(end)->AdjustEdge(1);
00630 
00631                 res = starter->FindEdge('D');
00632                 Assert::IsNull(res);
00633 
00634             }
00635         };
00636 
00637     }
00638 
00639     /** @brief Testing namespace for MarkovPasswords
00640     */
00641     namespace MarkovPasswords {
00642 
00643     };
00644 
00645 }
```

Index