phylotoy

0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 Controller Class Reference

`#include <Controller.h>`

Public Member Functions

- Controller ()
- void SetRandomSeed (int seed)
- int GetRandomSeed ()
- int CheckRandomSeed ()
- void SetNumberOfGenerations (int ngens)
- int CheckNumberOfGenerations ()
- void SetAlignmentFilePath (std::string path)
- std::string GetAlignmentFilePath ()
- std::string CheckAlignmentFilePath ()
- void SetChainName (std::string name)
- std::string GetChainName ()
- std::string CheckChainName ()
- void SetBLUniformMin (double u_min)
- double GetBLUniformMin ()
- void SetBLUniformMax (double u_max)
- double GetBLUniformMax ()
- void SetBLExponentialMean (double e_mean)
- double GetBLExponentialMean ()
- void CheckCLIOptions ()
- void Run ()

3.1.1 Detailed Description

Definition at line 15 of file Controller.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Controller()

```
Controller::Controller ( )
```

Definition at line 16 of file Controller.cpp.

```
16                      {
17
18 }
```

3.1.3 Member Function Documentation

3.1.3.1 CheckAlignmentFilePath()

```
std::string Controller::CheckAlignmentFilePath ( )
```

Checks that the user provided alignment path is not empty

Returns

a std::string with the alignment path or raises an exception

Definition at line 110 of file Controller.cpp.

```
110                                             {
111
112   //if the user forgot to set the ali path this char* will be null and we need to throw an exception
113   if (!alignment_file_path.length()){
114
115     throw "Alignment file path (option -i) was not set but is required";
116
117   }else{
118
119     return alignment_file_path;
120   }
121
122 }
```

3.1.3.2 CheckChainName()

```
std::string Controller::CheckChainName ( )
```

Check the user provided chain name is not empty

Returns

a std::string with the chain name or raises an exception

Definition at line 148 of file Controller.cpp.

```
148                                       {
149
150   //if the user forgot to set the chain name this char* will be null and we need to throw an exception
151   if (!chain_name.length()){
152
153     throw "Chain name (option -c) was not set but is required";
154
155   }else{
156
157     return chain_name;
158
159   }
160
161 }
```

3.1.3.3 CheckCLIOptions()

```
void Controller::CheckCLIOptions ( )
```

Checks that all options are fine

Definition at line 227 of file Controller.cpp.

```
227                                {
228
229   /*We need to check whether the necessary options were set*/
230
231   try {
232
233     this->CheckAlignmentFilePath();
234     this->CheckChainName();
235     this->CheckRandomSeed();
236     this->CheckNumberOfGenerations();
237
238   } catch (const char* exception) {
239
240     std::string what_exception(exception);
241     std::string error = "Error: " + what_exception + "\n";
242     output_printer.PrintMessage2Out(error);
243
244     exit(1);
245
246   }
247
248 }
```

3.1.3.4 CheckNumberOfGenerations()

```
int Controller::CheckNumberOfGenerations ( )
```

Checks the user provided a valid number of generations

Returns

the number of generations or raises and exception

Definition at line 73 of file Controller.cpp.

```
73                                        {
74   //if the user forgot to set the random seed this int will be null and we need to throw an exception
75   if (number_of_generations <= 0){
76
77     throw "Number of generations (option -g) was not set but is required";
78
79   }else{
80
81     return number_of_generations;
82   }
83
84 }
```

3.1.3.5 CheckRandomSeed()

```
int Controller::CheckRandomSeed ( )
```

Checks the user provided random see is valid

Returns

int rando_seed or raises an exception

Definition at line 45 of file Controller.cpp.

```
45                                {
46
47   //if the user forgot to set the random seed this int will be null and we need to throw an exception
48   if (random_seed <= 0){
49
50     throw "Random seed (option -r) was not set but is required";
51
52   }else{
53
54     return random_seed;
55   }
56
57 }
```

3.1.3.6 GetAlignmentFilePath()

```
std::string Controller::GetAlignmentFilePath ( )
```

Get the alignment path

Returns

a std::string with the user provided alignment path

Definition at line 100 of file Controller.cpp.

```
100                                          {
101
102     return alignment_file_path;
103
104 }
```

3.1.3.7 GetBLExponentialMean()

```
double Controller::GetBLExponentialMean ( )
```

Get the mean value of the exponential distribution used to get the branch lengths

Returns

double with the user provided mean value

Definition at line 217 of file Controller.cpp.

```
217                                       {
218
219   return bl_exponential_mean;
220
221 }
```

3.1.3.8 GetBLUniformMax()

```
double Controller::GetBLUniformMax ( )
```

Get the max value of the uniform distribution used to get the branch lengths

Returns

double with the user provided max value

Definition at line 197 of file Controller.cpp.

```
197                                  {
198
199   return bl_uniform_max;
200
201 }
```

3.1.3.9 GetBLUniformMin()

```
double Controller::GetBLUniformMin ( )
```

Get the min value of the uniform distribution used to get the branch lengths

Returns

double with the user provided min value

Definition at line 177 of file Controller.cpp.

```
177                                  {
178
179   return bl_uniform_min;
180
181 }
```

3.1.3.10 GetChainName()

```
std::string Controller::GetChainName ( )
```

Get the user provided chain name

Returns

std::string with the chain name

Definition at line 138 of file Controller.cpp.

```
138                                   {
139
140     return chain_name;
141
142 }
```

3.1.3.11 GetRandomSeed()

```
int Controller::GetRandomSeed ( )
```

Get the random seed

Returns

int the random seed

Definition at line 35 of file Controller.cpp.

```
35                                {
36
37   return random_seed;
38
39 }
```

3.1.3.12 Run()

```
void Controller::Run ( )
```

This method initiliases the tree object and starts the run.

Definition at line 255 of file Controller.cpp.

```
255                       {
256
257   this->CheckCLIOptions();
258
259   //if all option are set, we tell the user how the program was invoked.
260   std::string program_call = "phylotoy was invoked with the following options:\n\n\tRandom seed = " +
      std::to_string(random_seed) + "\n\tAlingment path = " + alignment_file_path + "\n\tChain name = " + chain_name +
      "\n\n";
261
262   output_printer.PrintMessage2Out(program_call);
263
264   //tmp vector of strings
265   std::vector<std::string>* alignment;
266
267   output_printer.PrintMessage2Out("reading alignment\n");
268   //now we need to open and store the Alignment in a string vector
269   alignment = input_reader.ReadInputFile(alignment_file_path);
270
271
272   output_printer.PrintMessage2Out("initializing tree\n");
273
274   int number_of_nodes = phylo_tree.CreateBifurcatingTree(alignment);
275
276   for(int i=0; i < number_of_generations; i++){
277
278     std::cerr << "Select Node " << distribution_sampler.SampleFromIntUniform(0,
      number_of_nodes) << "\n";
279     //std::cerr << "Sample BL  " << distribution_sampler.SampleBLFromUniform(bl_uniform_min,
       bl_uniform_max) << "\n";
280     std::cerr << "Sample BL  " << distribution_sampler.SampleFromExponential(
      bl_exponential_mean) << "\n";
281   }
282 }
```

3.1.3.13 SetAlignmentFilePath()

```
void Controller::SetAlignmentFilePath (
             std::string path )
```

Set the alignment file path

Parameters

path	an std::string object with the user provided path to the alignment

Definition at line 90 of file Controller.cpp.

```
90                                                  {
91
92   alignment_file_path = path;
93
94 }
```

3.1.3.14 SetBLExponentialMean()

```
void Controller::SetBLExponentialMean (
             double e_mean )
```

Set the mean value of the exponential distribution used to get the branch lengths

Parameters

e_mean	a double with the user provided mean value

Definition at line 207 of file Controller.cpp.

```
207                                                 {
208
209   bl_exponential_mean = e_mean;
210
211 }
```

3.1.3.15 SetBLUniformMax()

```
void Controller::SetBLUniformMax (
             double u_max )
```

Set the max value for the uniform distribution used to get the branch lengths

Parameters

double	with the user provided value

Definition at line 187 of file Controller.cpp.

```
187                                          {
188
```

```
189   bl_uniform_max = u_max;
190
191 }
```

3.1.3.16 SetBLUniformMin()

```
void Controller::SetBLUniformMin (
             double u_min )
```

Set the min value for the uniform distribution used to get the branch lengths

Parameters

double	with the user provided value

Definition at line 167 of file Controller.cpp.

```
167                                          {
168
169   bl_uniform_min = u_min;
170
171 }
```

3.1.3.17 SetChainName()

```
void Controller::SetChainName (
             std::string name )
```

Set the name of the name

Parameters

name	a std::string provided by the user

Definition at line 128 of file Controller.cpp.

```
128                                          {
129
130   chain_name = name;
131
132 }
```

3.1.3.18 SetNumberOfGenerations()

```
void Controller::SetNumberOfGenerations (
            int ngens )
```

Set the number of generations

Parameters

ngens	an integer providing the desired number of generations for the chain to run

Definition at line 63 of file Controller.cpp.

```
63                                                  {
64
65   number_of_generations = ngens;
66
67 }
```

3.1.3.19 SetRandomSeed()

```
void Controller::SetRandomSeed (
             int seed )
```

Set the random seed used for the number generator

Parameters

seed,an	integer especifying the random seed

Definition at line 24 of file Controller.cpp.

```
24                                          {
25
26   random_seed = seed;
27   distribution_sampler.SetRandomNumberGeneratorSeed(seed);
28
29 }
```

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

- /home/sergio/Repos/phylotoy/src/Controller.h
- /home/sergio/Repos/phylotoy/src/Controller.cpp

3.2 DistributionSampler Class Reference

```
#include <DistributionSampler.h>
```

Public Member Functions

- DistributionSampler ()
- void SetRandomNumberGeneratorSeed (int seed)
- int SampleFromIntUniform (int min, int max)
- double SampleFromRealUniform (double min, double max)
- double SampleFromExponential (double mean)
- double SampleFromGamma (double alpha, double beta)

3.2.1 Detailed Description

phylotoy Class DistributionSampler objects of this class provide access to different probabily distributions.

Definition at line 12 of file DistributionSampler.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 DistributionSampler()

```
DistributionSampler::DistributionSampler ( )
```

Class DistributionSampler

Definition at line 12 of file DistributionSampler.cpp.

```
12 {}
```

3.2.3 Member Function Documentation

3.2.3.1 SampleFromExponential()

```
double DistributionSampler::SampleFromExponential (
            double mean )
```

This method samples from an exponential distribution a real number

Definition at line 54 of file DistributionSampler.cpp.

```
54                                                          {
55
56   std::exponential_distribution<double> exponential_sampler(mean);
57
58   std::cerr << "sampling branch length from exponential distribution (" << mean << ")\n";
59
60   return exponential_sampler(pseudo_random_number_gen);
61
62 }
```

3.2.3.2 SampleFromGamma()

```
double DistributionSampler::SampleFromGamma (
            double alpha,
            double beta )
```

This method samples from a gamma distribution a real number

Definition at line 68 of file DistributionSampler.cpp.

```
68                                                                  {
69
70   std::gamma_distribution<double> gamma_sampler(alpha, beta);
71
72   std::cerr << "sampling branch length from exponential distribution (" << alpha << "," << beta << ")\n";
73
74   return gamma_sampler(pseudo_random_number_gen);
75
76 }
```

3.2.3.3 SampleFromIntUniform()

```
int DistributionSampler::SampleFromIntUniform (
            int min,
            int max )
```

This method samples from a uniform distribution an integer number

Definition at line 27 of file DistributionSampler.cpp.

```
27                                                          {
28
29   std::uniform_int_distribution<int> uniform_int_sampler(min, max);
30
31   std::cerr << "sampling branch length from real uniform distribution (" << min << "," << max << ")\n";
32
33   return uniform_int_sampler(pseudo_random_number_gen);
34
35 }
```

3.2.3.4 SampleFromRealUniform()

```
double DistributionSampler::SampleFromRealUniform (
            double min,
            double max )
```

This method samples from a uniform distribution a real number

Definition at line 41 of file DistributionSampler.cpp.

```
41                                                              {
42
43   std::uniform_real_distribution<double> uniform_real_sampler(min, max);
44
45   std::cerr << "sampling branch length from real uniform distribution (" << min << "," << max << ")\n";
46
47   return uniform_real_sampler(pseudo_random_number_gen);
48
49 }
```

3.2.3.5 SetRandomNumberGeneratorSeed()

```
void DistributionSampler::SetRandomNumberGeneratorSeed (
            int seed )
```

Definition at line 15 of file DistributionSampler.cpp.

```
15                                                              {
16
17   std::cerr << "setting random seed = " << seed <<" for pseudo_random_number_gen\n";
18   pseudo_random_number_gen.seed(seed);
19
20 }
```

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

- /home/sergio/Repos/phylotoy/src/DistributionSampler.h
- /home/sergio/Repos/phylotoy/src/DistributionSampler.cpp

3.3 InputReader Class Reference

```
#include <InputReader.h>
```

Public Member Functions

- InputReader ()
- InputReader (std::string path)
- void SetPath (std::string path)
- std::string GetPath ()
- std::vector< std::string > * ReadInputFile (std::string path)

3.3.1 Detailed Description

Definition at line 11 of file InputReader.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 InputReader() [1/2]

```
InputReader::InputReader ( )
```

Definition at line 13 of file InputReader.cpp.

```
13 {}
```

3.3.2.2 InputReader() [2/2]

```
InputReader::InputReader (
            std::string path )
```

Definition at line 15 of file InputReader.cpp.

```
15                                      {
16
17   input_file_path = path;
18
19 }
```

3.3.3 Member Function Documentation

3.3.3.1 GetPath()

```
std::string InputReader::GetPath ( )
```

Definition at line 27 of file InputReader.cpp.

```
27                                {
28
29   return input_file_path;
30
31 }
```

3.3.3.2 ReadInputFile()

```
std::vector< std::string > * InputReader::ReadInputFile (
            std::string path )
```

Definition at line 33 of file InputReader.cpp.

```
33                                                              {
34
35   /* create and open the input stream. This will always be done, we need to control later if the stream
36    * is good or not...
37   */
38
39   std::ifstream input_stream(path);
40
41   //create a vector of strings to store the data
42   std::vector<std::string>* lines = new std::vector<std::string>;
43
44   if(input_stream.good()) {
45
46     //dummy string to store the first line
47     std::string first_line;
48
49     //read the first line
50     std::getline(input_stream, first_line);
51
52     //string to store the lines.
53     std::string input_line;
54
55     //now read the file line by line and push the line into the vector
```

```
56     while(std::getline(input_stream, input_line)) {
57
58       //add the lines to the string vector
59       lines->push_back(input_line);
60
61     }
62
63     //input_stream.close();
64
65   }else {
66
67     std::cerr << "Something went wrong reading the alignment file: " << path << "\n";
68     exit(1);
69
70   }
71
72
73   return lines;
74
75 }
```

3.3.3.3 SetPath()

```
void InputReader::SetPath (
              std::string path )
```

Definition at line 21 of file InputReader.cpp.

```
21                                        {
22
23   input_file_path = path;
24
25 }
```

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

- /home/sergio/Repos/phylotoy/src/InputReader.h
- /home/sergio/Repos/phylotoy/src/InputReader.cpp

3.4 Node Class Reference

```
#include <Node.h>
```

Public Member Functions

- Node ()
- void SetSequence (std::string species_sequence)
- std::string GetSequence ()
- void SetSpeciesName (std::string name)
- std::string GetSpeciesName ()
- void SetIndex (int node_index)
- int GetIndex ()
- void SetIsTip (bool tip)
- bool GetIsTip ()
- void SetParentNode (Node *parent)

- Node * GetParentNode ()
- void SetLengthSubtendingBranch (float branch_length)
- float GetLengthSubtendingBranch ()
- void AddNodeToChildVector (Node *child)
- void SetChildVector (std::vector< Node *> childs)
- std::vector< Node * > GetChildVector ()
- void CreateBifurcatingNode (std::vector< Node *>, int &calls, std::vector< Node *> &tree_nodes)
- void GetNodePointer (std::vector< Node *> *tree_nodes)
- std::vector< std::string > * GetNodeInfo (std::vector< std::string > *collected_node_info)
- void GetNodeInfoInNewickFormat (std::string &newick_tree)

3.4.1 Detailed Description

Definition at line 11 of file Node.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Node()

```
Node::Node (
            void  )
```

Definition at line 14 of file Node.cpp.

```
14 {}
```

3.4.3 Member Function Documentation

3.4.3.1 AddNodeToChildVector()

```
void Node::AddNodeToChildVector (
            Node * child )
```

Definition at line 92 of file Node.cpp.

```
92                                      {
93
94   child_nodes.push_back(child);
95
96 }
```

3.4.3.2 CreateBifurcatingNode()

```
void Node::CreateBifurcatingNode (
             std::vector< Node *> tip_nodes,
             int & current_node_index,
             std::vector< Node *> & tree_nodes )
```

We create the tree as follows: the current node has no children, i.e. the size of the vector child_nodes is 0. We insert the tip node as a child of this node, sending a reference of it to the child node for it to have a pointer to its parent node. Once this is done, we create a new internode, set this node as its parent and add it to the child_nodes vector of the current node. Then we recursively call this method on the new internode but only if we still have more than 1 tip nodes left. If not, we add the last tip node to the current internode.

After this happens the vector of tip nodes should be empty and the method returns.

Parameters

tip_nodes	a std::vector of Node pointers containing the tip nodes
current_node_index	an int pointer to be used to allocate each node with an index.
tree_nodes	a pointer to a std:vector of Node pointers that contains all nodes in a tree.

if only one last tip node is left. We can add it to the current internode. This internode should only have 1 descendent tip.

Definition at line 128 of file Node.cpp.

```
128
          {
129
130   std::cerr << "creating bifurcating node " << tip_nodes.size() << "\n";
131
132   if(!tip_nodes.empty()){
133
134     Node* tip_node_to_insert = tip_nodes.back();
135     tip_nodes.pop_back();//this deletes the last element of the array
136
137     std::cerr << "inserting tip: " << tip_node_to_insert->GetSpeciesName() << "\n";
138     tip_node_to_insert->SetParentNode(this);
139     this->AddNodeToChildVector(tip_node_to_insert);
140     tree_nodes.push_back(tip_node_to_insert);
141
142     if(tip_nodes.size() == 1){
143
150       tip_node_to_insert = tip_nodes.back();
151       tip_nodes.pop_back();//this deletes the last element of the array
152
153       std::cerr << "inserting last tip: " << tip_node_to_insert->GetSpeciesName() << "\n";
154
155       tip_node_to_insert->SetParentNode(this);
156       this->AddNodeToChildVector(tip_node_to_insert);
157       tree_nodes.push_back(tip_node_to_insert);
158
159     }else{
160
161       std::cerr << "inserting new internode " << current_node_index << "\n";
162
163       Node* internode = new Node();
164       internode->SetIsTip(false);
165       internode->SetParentNode(this);
166       internode->SetIndex(current_node_index);
167       this->AddNodeToChildVector(internode);
168       tree_nodes.push_back(internode);
169
170       current_node_index = current_node_index + 1;//increase the index counter
171       internode->CreateBifurcatingNode(tip_nodes, current_node_index, tree_nodes);
172
173     }
174   }
175 }
```

3.4.3.3 GetChildVector()

```
std::vector< Node * > Node::GetChildVector ( )
```

Definition at line 104 of file Node.cpp.

```
104                                     {
105
106   return child_nodes;
107
108 }
```

3.4.3.4 GetIndex()

```
int Node::GetIndex ( )
```

Definition at line 60 of file Node.cpp.

```
60                  {
61
62   return index;
63
64 }
```

3.4.3.5 GetIsTip()

```
bool Node::GetIsTip ( )
```

Definition at line 48 of file Node.cpp.

```
48                   {
49
50   return is_tip;
51
52 }
```

3.4.3.6 GetLengthSubtendingBranch()

```
float Node::GetLengthSubtendingBranch ( )
```

Definition at line 73 of file Node.cpp.

```
73                                      {
74
75   return length_of_subtending_branch;
76
77 }
```

3.4.3.7 GetNodeInfo()

```
std::vector< std::string > * Node::GetNodeInfo (
            std::vector< std::string > * collected_node_info )
```

Definition at line 210 of file Node.cpp.

```
210                                                                                 {
211
212   if(child_nodes.empty()) {
213
214     std::cerr << "empty child node vector\n";
215
216     //add the tip node index, species name and sequence to the vector collecting the node information
217     std::string node_info = std::to_string(index) + ' ' + species_name + ' ' + sequence;
218
219     std::cerr << "adding " << node_info << " to info vector\n";
220
221     collected_node_info->push_back(node_info);
222
223
224   }else {
225
226     std::cerr << "recursively calling nodes " << child_nodes.size() << "\n";
227     //for each child node, call this function.
228     for(auto child : child_nodes) {
229
230       std::cerr << "in for\n";
231       child->GetNodeInfo(collected_node_info);
232
233     }
234
235     std::cerr << "back at internode \n";
236
237     //add internode index to the vector collecting the node information
238     std::string node_info = std::to_string(index);
239
240     std::cerr << "adding " << node_info << " to info vector\n";
241
242     collected_node_info->push_back(node_info);
243
244
245   }
246
247   return collected_node_info;
248
249 }
```

3.4.3.8 GetNodeInfoInNewickFormat()

```
void Node::GetNodeInfoInNewickFormat (
            std::string & newick_tree )
```

This method modifies a

Parameters

newick_tree	a reference to std::string for the Node currently executing the method to store his information in newick before calling the method on its children Nodes.

Returns

void: the method directly modifies the string.

for each child node, call this function. Note that because we do not have left and right child nodes but a vector of Node objects, and we want this to be able to implemente polytomies later, we need to check whether the node object at hand is the last of the iterator to write a , or not.

Definition at line 259 of file Node.cpp.

```
259                                                     {
260
261   if(child_nodes.empty()){
262
263     newick_tree.append(species_name);
264
265   }else {
266
267     newick_tree.append("(");
268
276     for(auto child : child_nodes) {
277
278       child->GetNodeInfoInNewickFormat(newick_tree);
279
280       if(child->GetIsTip() && child != child_nodes.back()){
281
282         newick_tree.append(",");
283
284       }else {
285
286         newick_tree.append(")");
287
288       }
289     }
290   }
291 }
```

3.4.3.9 GetNodePointer()

```
void Node::GetNodePointer (
             std::vector< Node *> * tree_nodes )
```

Definition at line 178 of file Node.cpp.

```
178                                                 {
179
180   if(child_nodes.empty()) {
181
182     std::cerr << "empty child node vector\n";
183
184     std::cerr << "adding node " << index << " to node ref vector\n";
185
186     tree_nodes->push_back(this);
187
188   }else {
189
190     std::cerr << "recursively calling nodes" << child_nodes.size() << "\n";
191     //for each child node, call this function.
192     for(auto child : child_nodes) {
193
194       std::cerr << "in for\n";
195       child->GetNodePointer(tree_nodes);
196
197     }
198
199     std::cerr << "back at internode \n";
200
201     std::cerr << "adding node " << index << " to node ref vector\n";
202
203     tree_nodes->push_back(this);
204
205   }
206
207 }
```

3.4.3.10 GetParentNode()

```
Node * Node::GetParentNode ( )
```

Definition at line 85 of file Node.cpp.

```
85                          {
86
87   return parent_node;
88
89 }
```

3.4.3.11 GetSequence()

```
std::string Node::GetSequence ( )
```

Definition at line 23 of file Node.cpp.

```
23                            {
24
25   return sequence;
26
27 }
```

3.4.3.12 GetSpeciesName()

```
std::string Node::GetSpeciesName ( )
```

Definition at line 36 of file Node.cpp.

```
36                               {
37
38   return species_name;
39
40 }
```

3.4.3.13 SetChildVector()

```
void Node::SetChildVector (
             std::vector< Node *> childs )
```

Definition at line 98 of file Node.cpp.

```
98                                                {
99
100   child_nodes = childs;
101
102 }
```

3.4.3.14 SetIndex()

```
void Node::SetIndex (
            int node_index )
```

Definition at line 54 of file Node.cpp.

```
54                                {
55
56   index = node_index;
57
58 }
```

3.4.3.15 SetIsTip()

```
void Node::SetIsTip (
            bool tip )
```

Definition at line 42 of file Node.cpp.

```
42                           {
43
44   is_tip = tip;
45
46 }
```

3.4.3.16 SetLengthSubtendingBranch()

```
void Node::SetLengthSubtendingBranch (
            float branch_length )
```

Definition at line 67 of file Node.cpp.

```
67                                                      {
68
69   length_of_subtending_branch = branch_length;
70
71 }
```

3.4.3.17 SetParentNode()

```
void Node::SetParentNode (
            Node * parent )
```

Definition at line 79 of file Node.cpp.

```
79                                  {
80
81   parent_node = parent;
82
83 }
```

3.4.3.18 SetSequence()

```
void Node::SetSequence (
            std::string species_sequence )
```

Definition at line 16 of file Node.cpp.

```
16                                                  {
17
18   std::cerr << "setting sequence\n";
19   sequence = species_sequence;
20
21 }
```

3.4.3.19 SetSpeciesName()

```
void Node::SetSpeciesName (
            std::string name )
```

Definition at line 29 of file Node.cpp.

```
29                                        {
30
31   std::cerr << "setting species name " << name <<"\n";
32   species_name = name;
33
34 }
```

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

- /home/sergio/Repos/phylotoy/src/Node.h
- /home/sergio/Repos/phylotoy/src/Node.cpp

3.5 OutputPrinter Class Reference

`#include <OutputPrinter.h>`

Public Member Functions

- OutputPrinter ()
- void PrintMessage2Out (std::string text)

3.5.1 Detailed Description

Definition at line 10 of file OutputPrinter.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 OutputPrinter()

```
OutputPrinter::OutputPrinter ( )
```

Definition at line 11 of file OutputPrinter.cpp.

```
11 {}
```

3.5.3 Member Function Documentation

3.5.3.1 PrintMessage2Out()

```
void OutputPrinter::PrintMessage2Out (
            std::string text )
```

Definition at line 13 of file OutputPrinter.cpp.

```
13                                                  {
14
15   std::cout << text;
16
17 }
```

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

- /home/sergio/Repos/phylotoy/src/OutputPrinter.h
- /home/sergio/Repos/phylotoy/src/OutputPrinter.cpp

3.6 Tree Class Reference

```
#include <Tree.h>
```

Public Member Functions

- Tree ()
- void SetLength (float length)
- float GetLength ()
- void SetRoot (Node *root_node)
- Node * GetRoot ()
- void SetTreeNodes (std::vector< Node *> nodes)
- std::vector< Node * > GetTreeNodes ()
- void AddToNodeVector (Node *node)
- void GetTreeNodeVector ()
- int CreateStarTree (std::vector< std::string > *alignment)
- int CreateBifurcatingTree (std::vector< std::string > *alignment)
- std::vector< std::string > * CollectTreeNodesInfoRecursively ()
- std::vector< std::string > * CollectTreeNodesInfoIteratively ()
- std::string GetTreeInNewickFormat ()

3.6.1 Detailed Description

Objects of the class Tree represent phylogenetic trees. These trees are (1) unrooted (the root node used in the code is just an internode of the tree, (2) can be initialized as start trees, (3) can contain polytomies, or (4) can be bifurcating.

This class provide a number of methods to manipulate the tree. For instance, re-rooting the tree. It also provides methods to alter the tree topology using Nearest-neighbor interchange (NNI) or subtree pruning and regraphting (SPR).

A number of attributes of the tree provide short-cuts to make modifying the tree topology or the length of the tree's branches easy. For instance, Tree objects store pointers to all of their nodes and to the current root. This makes easy to rearrange the tree during MCMC.

The Tree Class is also in charge of proposing all modifications to tree topology and branch length.

Definition at line 20 of file Tree.h.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 Tree()

```
Tree::Tree ( )
```

Definition at line 12 of file Tree.cpp.

```
12 {}
```

3.6.3 Member Function Documentation

3.6.3.1 AddToNodeVector()

```
void Tree::AddToNodeVector (
            Node * node )
```

Definition at line 52 of file Tree.cpp.

```
52                                      {
53
54   std::cerr << "here\n";
55   tree_nodes.push_back(node);
56
57 }
```

3.6.3.2 CollectTreeNodesInfoIteratively()

```
std::vector< std::string > * Tree::CollectTreeNodesInfoIteratively ( )
```

This method returns information on the nodes currently included in the tree. It used the tree_nodes vector of Node pointer to iteratively retrieve the Node information

Returns

a vector of strings

Definition at line 149 of file Tree.cpp.

```
149                                                         {
150
151   std::vector<std::string>* tree_nodes_info = new std::vector<std::string>;
152
153   std::cerr << "iterating over " << tree_nodes.size() << " tree nodes to get their information\n";
154
155   for (auto node : tree_nodes){
156
157     if(node->GetIsTip()){
158
159       //node is a tip, get the species name and sequence
160       std::string node_info = std::to_string(node->GetIndex()) + ' ' + node->GetSpeciesName() + ' ' + node
      ->GetSequence();
161       tree_nodes_info->push_back(node_info);
162       std::cerr << "adding tip: " << node_info << " to information vector\n";
163
164     }else{
165       //node is an internode get the index only
166       std::string node_info = std::to_string(node->GetIndex());
167       tree_nodes_info->push_back(node_info);
168       std::cerr << "adding internode: " << node_info << " to information vector\n";
169
170     }
171
172   }
173
174   return tree_nodes_info;
175 }
```

3.6.3.3 CollectTreeNodesInfoRecursively()

```
std::vector< std::string > * Tree::CollectTreeNodesInfoRecursively ( )
```

This method returns information on the nodes currently included in the tree. It recursively traverses the tree starting from the root node and asks each node for its info as a string that is stored on a vector of strings.

Returns

a vector of strings

Definition at line 133 of file Tree.cpp.

```
133                                                         {
134
135   std::vector<std::string>* tree_nodes_info = new std::vector<std::string>;
136
137   std::cerr << "at tree root\n";
138   tree_nodes_info = current_root->GetNodeInfo(tree_nodes_info);
139
140   return tree_nodes_info;
141 }
```

3.6.3.4 CreateBifurcatingTree()

```
int Tree::CreateBifurcatingTree (
            std::vector< std::string > * alignment )
```

This method creates a bifurcating tree. This method creates as many internodes as required to yield a bifurcating tree.

Parameters

alignment	is a sequence alignment stored on a vector of strings containing species names and sequencesseparated by an empty space.

Definition at line 101 of file Tree.cpp.

```
101                                                          {
102
103   int current_node_index = alignment->size();
104
105   //we need to initialize the root node
106   Node* current_root = new Node();
107   current_root->SetIsTip(false);
108   current_root->SetIndex(current_node_index);
109
110   //once the root node has been initialized we add it as the tree root and to the list of tree nodes
111   this->SetRoot(current_root);
112   this->AddToNodeVector(current_root);
113
114   std::cerr << "Creating bifurcating tree with " << current_node_index << " species\n";
115
116   current_node_index = current_node_index + 1;
117
118   //we initialize the tip nodes using the alignment provided by the user
119   current_root->CreateBifurcatingNode(this->InitializeTipNodes(alignment),
      current_node_index, tree_nodes);
120
121   //return the total number of nodes in the tree
122   return current_node_index;
123
124 }
```

3.6.3.5 CreateStarTree()

```
int Tree::CreateStarTree (
            std::vector< std::string > * alignment )
```

This method creates a star tree. This tree adds all the tip nodes in one alignment to the root node.

Parameters

alignment	is a sequence alignment with species names and sequences separated by an empty space.

Definition at line 64 of file Tree.cpp.

```
64                                                  {
65
66   std::cerr << "Creating star tree\n";
67
```

```
68   //initialize the root node
69   Node* current_root = new Node();
70   current_root->SetIsTip(false);
71   current_root->SetIndex(alignment->size());
72
73   //send the alignment to our private tip node initializer
74   current_root->SetChildVector(this->InitializeTipNodes(alignment));
75
76   //once all tips have been added as childs to the root, the root sends a pointer to himself to each of the
       child nodes.
77   for(auto child : current_root->GetChildVector()){
78
79     child->SetParentNode(current_root);
80     //we also need to set the length of the subtending branch leading to the parent
81
82     //finally we need to add this child to the tree list of nodes
83     this->AddToNodeVector(child);
84   }
85
86   //set current_root node as the root of the tree object
87   this->SetRoot(current_root);
88
89   //add root to the node vector
90   this->AddToNodeVector(current_root);
91
92   return current_root->GetIndex();
93
94 }
```

3.6.3.6 GetLength()

```
float Tree::GetLength ( )
```

Definition at line 20 of file Tree.cpp.

```
20                      {
21
22   return length;
23
24 }
```

3.6.3.7 GetRoot()

```
Node * Tree::GetRoot ( )
```

Definition at line 33 of file Tree.cpp.

```
33                     {
34
35   return current_root;
36
37 }
```

3.6.3.8 GetTreeInNewickFormat()

```
std::string Tree::GetTreeInNewickFormat ( )
```

This methods recursively calls every Node in the tree and produces a string with the tree in newick format.

Returns

a string object with the tree in newick format.

Definition at line 196 of file Tree.cpp.

```
196                                     {
197
198   std::string newick_tree;
199
200   current_root->GetNodeInfoInNewickFormat(newick_tree);
201
202   newick_tree.append(";");
203
204   return newick_tree;
205
206 }
```

3.6.3.9 GetTreeNodes()

```
std::vector< Node * > Tree::GetTreeNodes ( )
```

Definition at line 45 of file Tree.cpp.

```
45                                    {
46
47   return tree_nodes;
48
49 }
```

3.6.3.10 GetTreeNodeVector()

```
void Tree::GetTreeNodeVector ( )
```

In case we need to update the list of Nodes in the tree_nodes list, this method should provide a way to get pointers to all the nodes in the tree recursively

Definition at line 183 of file Tree.cpp.

```
183                             {
184
185   std::cerr << "at tree root\n";
186   current_root->GetNodePointer(&tree_nodes);
187
188 }
```

3.6.3.11 SetLength()

```
void Tree::SetLength (
            float length )
```

Definition at line 14 of file Tree.cpp.

```
14                                {
15
16     length = length;
17
18 }
```

3.6.3.12 SetRoot()

```
void Tree::SetRoot (
            Node * root_node )
```

Definition at line 26 of file Tree.cpp.

```
26                                 {
27
28   current_root = root_node;
29
30 }
```

3.6.3.13 SetTreeNodes()

```
void Tree::SetTreeNodes (
            std::vector< Node *> nodes )
```

Definition at line 39 of file Tree.cpp.

```
39                                            {
40
41   tree_nodes = nodes;
42
43 }
```

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

- /home/sergio/Repos/phylotoy/src/Tree.h
- /home/sergio/Repos/phylotoy/src/Tree.cpp

Chapter 4

File Documentation

4.1 /home/sergio/Repos/phylotoy/src/Controller.cpp File Reference

```
#include "Controller.h"
#include <string>
#include <exception>
#include <assert.h>
#include <vector>
#include <iostream>
```

4.2 /home/sergio/Repos/phylotoy/src/Controller.h File Reference

```
#include "InputReader.h"
#include "OutputPrinter.h"
#include "Tree.h"
#include "DistributionSampler.h"
#include <string>
#include <vector>
```

Classes

- class Controller

4.3 /home/sergio/Repos/phylotoy/src/DistributionSampler.cpp File Reference

```
#include "DistributionSampler.h"
#include <string>
#include <vector>
#include <iostream>
```

4.4 /home/sergio/Repos/phylotoy/src/DistributionSampler.h File Reference

```
#include <string>
#include <vector>
#include <random>
```

Classes

- class DistributionSampler

4.5 /home/sergio/Repos/phylotoy/src/InputReader.cpp File Reference

```
#include "InputReader.h"
#include <fstream>
#include <vector>
#include <iostream>
```

4.6 /home/sergio/Repos/phylotoy/src/InputReader.h File Reference

```
#include <vector>
#include <string>
```

Classes

- class InputReader

4.7 /home/sergio/Repos/phylotoy/src/Node.cpp File Reference

```
#include <vector>
#include <string>
#include "Node.h"
#include <iostream>
```

4.8 /home/sergio/Repos/phylotoy/src/Node.h File Reference

```
#include <vector>
#include <string>
```

Classes

- class Node

4.9 /home/sergio/Repos/phylotoy/src/OutputPrinter.cpp File Reference

```
#include <iostream>
#include "OutputPrinter.h"
```

4.10 /home/sergio/Repos/phylotoy/src/OutputPrinter.h File Reference

```
#include <string>
```

Classes

- class OutputPrinter

4.11 /home/sergio/Repos/phylotoy/src/Phylotoy.cpp File Reference

```
#include <getopt.h>
#include <stdlib.h>
#include "Controller.h"
#include <string.h>
#include <iostream>
```

Functions

- int main (int argc, char ∗argv[ ])

4.11.1 Function Documentation

4.11.1.1 main()

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

phylotoy

Author

Sergio Vargas

A toy program to learn how bayesian phylogenetic reconstruction works... Read the CLI options set the appropriate variables in the controller Options are passsed as follows: –random_seed expects an integer that will be used as the random seed for the run –input_file expect a path to the alignment file –chain_name expects the user to provide a name for the chain –generations expects an integer especifying the number of generations –uniform_branches Branch lengths will be sampled from a Uniform distribution. Expects two floats separated by a , and especifying the min and max values for the uniform branch length distribution. –exponential_branches expects a float especifying the mean value of the exponential distribution –uniform_dirichlet_branches Branch lengths will be sampled from a uniform Dirichlet distribution. No argument expected. –non_uniform_dirichlet_branches Branch lengths will be sampled from a non-uniform Dirichlet distribution. No argument expected. –verbose activate verbosity

Definition at line 16 of file Phylotoy.cpp.

```
16                                {
17
18  //Initialize a controller object
19
20  Controller phylotoy_controller;
21
37  int option;
38
39  static struct option long_options[] = {{ "random_seed",  required_argument, 0, 'r'},
40                                         { "input_file", required_argument, 0, 'i'},
41                                         { "chain_name", required_argument, 0, 'c'},
42                                         { "generations", required_argument, 0, 'g'},
43                                         { "uniform_branches", required_argument, 0, 'u'},
44                                         { "exponential_branches", required_argument, 0, 'e'},
45                                         { "uniform_dirichlet_branches", required_argument, 0, 'f'},
46                                         { "non_uniform_dirichlet_branches", required_argument, 0, 'o'},
47                                         { "verbose", no_argument, 0, 'v'},
48                                         {0, 0, 0, 0}
49  };
50
51  while ((option = getopt(argc, argv, "r:i:c:g:u:e:ud:nd:v")) != -1) {
52
53    switch (option){
54      case 'r':
55      {
56        phylotoy_controller.SetRandomSeed(atoi(optarg));
57        break;
58      }
59      case 'i':
60      {
61        std::string input (optarg);
62        phylotoy_controller.SetAlignmentFilePath(input);
63        break;
64      }
65      case 'c':
66      {
67        std::string name (optarg);
68        phylotoy_controller.SetChainName(name);
69        break;
70      }
71      case 'g':
72      {
73        phylotoy_controller.SetNumberOfGenerations(atoi(optarg));
74        break;
75      }
76      case 'u':
77      {
78        char* min = strtok(optarg, ",");
79        char* max = strtok(NULL, ",");
```

```
80          phylotoy_controller.SetBLUniformMin(atoi(min));
81          phylotoy_controller.SetBLUniformMax(atoi(max));
82          break;
83        }
84        case 'e':
85        {
86          phylotoy_controller.SetBLExponentialMean(atof(optarg));
87          break;
88        }
89        case 'f':
90        {
91          break;
92
93        }
94        case 'o':
95        {
96          break;
97
98        }
99        default:
100         abort();
101     }
102   }
103
104   phylotoy_controller.Run();
105
106   return 0;
107 }
```

4.12 /home/sergio/Repos/phylotoy/src/Tree.cpp File Reference

```
#include <vector>
#include <string>
#include "Tree.h"
#include <sstream>
#include <iostream>
```

4.13 /home/sergio/Repos/phylotoy/src/Tree.h File Reference

```
#include <vector>
#include <string>
#include "Node.h"
```

Classes

- class Tree

Index