TreeNode

1.0.0

Generated by Doxygen 1.8.18

Chapter 1

TreeNode C# library

This is the documentation website for the **TreeNode C# library**.

TreeNode is a library for reading, writing and manipulating phylogenetic trees in C# and R. It can open and create files in the most common phylogenetic formats (i.e. Newick/New Hampshire and NEXUS) and adds support for two new formats, the `Newick-with-Attributes` and `Binary format`.

The **TreeNode C# library**, in addition to providing methods to read and write phylogenetic tree files, also includes methods to manipulate the resulting trees (e.g. to reroot the tree, compute a consensus tree, find the last common ancestor of a group, etc.).

TreeNode is released under the `GPLv3` licence.

1.1 Getting started

The TreeNode C# library targets .NET Standard 2.1, thus it can be used in projects that target .NET Standard 2.1+ and .NET Core 3.0+, as well as Mono and Xamarin.

To use the library in your project, you should install the `TreeNode NuGet package`.

1.2 Usage

The `Examples` project in the `TreeNode GitHub repository` contains an example C# .NET Core console program showing some of the capabilities of the library.

The `PhyloTree` namespace contains the `TreeNode` class, which is used to represent nodes in a tree. `TreeNode` does not distinguish between internal nodes, tips or even whole trees (except when looking at some specific properties - e.g. a tip will not have any `Children`, and the root node of the tree will not have any `Parent`). This makes it possible to navigate the tree in an intuitive manner: for example, the ancestor of a `TreeNode` can be accessed using its `Parent` property (which is itself a `TreeNode`) and the descendants of a node can be found as the node's `Children` (which is a `List<TreeNode>`).

A full list of the information that can be extracted and the manipulations that can be performed on `TreeNode` objects can be obtained by looking at the methods and properties of the `TreeNode` class in this website.

In addition to this, the PhyloTree namespace contains the Phylotree.Formats namespace. The three classes in this namespace (NWKA, NEXUS and BinaryTree) contain methods that can be used to read and write TreeNode objects to files in the respective format.

Each of these classes offers (at least) the following methods (with additional optional arguments):

```
//Methods to read trees
IEnumerable<TreeNode> ParseTrees(string inputFile);
IEnumerable<TreeNode> ParseTrees(Stream inputStream);
List<TreeNode> ParseAllTrees(string inputFile);
List<TreeNode> ParseAllTrees(Stream inputStream);
//Methods to write trees
void WriteTree(TreeNode tree, string outputFile);
void WriteTree(TreeNode tree, Stream outputStream);
void WriteAllTrees(IEnumerable<TreeNode> trees, string outputFile);
void WriteAllTrees(IEnumerable<TreeNode> trees, Stream outputStream);
void WriteAllTrees(List<TreeNode> trees, string outputFile);
void WriteAllTrees(List<TreeNode> trees, Stream outputStream);
```

The ParseTrees methods can be used to read trees off a file or a Stream, without having to load them completely into memory. This can be useful if each tree only needs to be processed briefly. The ParseAllTrees methods instead load all the trees from the file into memory.

The WriteTree methods are used to write a single tree to a file or a stream, while the WriteAllTrees methods write a collection of trees.

In addition to this, the library also provides the TreeCollection class, which represents a collection of trees, much like a List<TreeNode>. However, a TreeCollection can also be created by passing a stream of trees in binary format to it: in this case, the TreeCollection will only parse trees from the stream when necessary, thus reducing the amount of memory that is necessary to store them.

The key feature of TreeCollection is that this is done *transparently*: accessing an element of the collection, e.g. by using treeCollection[i], will automatically perform all the reading and parsing operations from the stream to produce the TreeNode that is returned. This makes it possible to have an "agnostic" interface that behaves in the same way whether the trees in the collection have been completely loaded into memory or not.

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Namespace Documentation

5.1 PhyloTree Namespace Reference

Contains classes and methods to read, write and manipulate phylogenetic trees.

Namespaces

- namespace Extensions

  Contains useful extension methods.
- namespace Formats

  Contains classes and methods to read and write phylogenetic trees in multiple formats

Classes

- class AttributeDictionary

  Represents the attributes of a node. Attributes Name, Length and Support are always included. See the respective properties for default values.
- class TreeCollection

  Represents a collection of TreeNode objects. If the full representations of the TreeNode objects reside in memory, this offers the best performance at the expense of memory usage. Alternatively, the trees may be read on demand from a stream in binary format. In this case, accessing any of the trees will require the tree to be parsed. This reduces memory usage, but worsens performance. The internal storage model of the collection is transparent to consumers (except for the difference in performance/memory usage).
- class TreeNode

  Represents a node in a tree (or a whole tree).

5.1.1 Detailed Description

Contains classes and methods to read, write and manipulate phylogenetic trees.

5.2 PhyloTree.Extensions Namespace Reference

Contains useful extension methods.

Classes

- class TypeExtensions

  Useful extension methods

5.2.1 Detailed Description

Contains useful extension methods.

5.3 PhyloTree.Formats Namespace Reference

Contains classes and methods to read and write phylogenetic trees in multiple formats

Classes

- struct Attribute

  Describes an attribute of a node.
- class BinaryTree

  Contains methods to read and write tree files in binary format.
- class BinaryTreeMetadata

  Holds metadata information about a file containing trees in binary format.
- class NEXUS

  Contains methods to read and write trees in NEXUS format.
- class NWKA

  Contains methods to read and write trees in Newick and Newick-with-Attributes (NWKA) format.

5.3.1 Detailed Description

Contains classes and methods to read and write phylogenetic trees in multiple formats

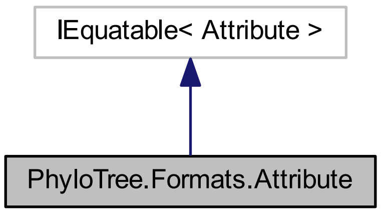
Chapter 6

Class Documentation

6.1 PhyloTree.Formats.Attribute Struct Reference

Describes an attribute of a node.

Inheritance diagram for PhyloTree.Formats.Attribute:



Public Member Functions

- Attribute (string attributeName, bool isNumeric)

  Constructs a new Attribute.
- override bool Equals (object obj)

  Compares an Attribute and another object.
- override int GetHashCode ()

  Returns the hash code for this Attribute.
- bool Equals (Attribute other)

  Compares two Attributes.

Static Public Member Functions

- static bool operator== (Attribute left, Attribute right)

  Compares two Attributes.
- static bool operator!= (Attribute left, Attribute right)

  Compares two Attributes (negated).

Properties

- string AttributeName `[get]`

  The name of the attribute.
- bool IsNumeric `[get]`

  Whether the attribute is represented by a numeric value or a string.

6.1.1 Detailed Description

Describes an attribute of a node.

Definition at line 692 of file Binary.cs.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Attribute()

```
PhyloTree.Formats.Attribute.Attribute (
             string attributeName,
             bool isNumeric )
```

Constructs a new Attribute.

Parameters

attributeName	The name of the attribute.
isNumeric	Whether the attribute is represented by a numeric value or a string.

Definition at line 709 of file Binary.cs.

6.1.3 Member Function Documentation

6.1.3.1 Equals() `[1/2]`

```
bool PhyloTree.Formats.Attribute.Equals (
             Attribute other )
```

Compares two Attributes.

Parameters

other	The Attribute to compare to.

Returns

`true` if *other* has the same AttributeName (case insensitive) and value for IsNumeric as the current instance. `false` otherwise.

Definition at line 768 of file Binary.cs.

6.1.3.2 Equals() `[2/2]`

```
override bool PhyloTree.Formats.Attribute.Equals (
            object obj )
```

Compares an Attribute and another object.

Parameters

obj	The object to compare to.

Returns

`true` if *obj* is an Attribute and it has the same AttributeName (case insensitive) and value for IsNumeric as the current instance. `false` otherwise.

Definition at line 720 of file Binary.cs.

6.1.3.3 GetHashCode()

```
override int PhyloTree.Formats.Attribute.GetHashCode ( )
```

Returns the hash code for this Attribute.

Returns

The hash code for this Attribute.

Definition at line 736 of file Binary.cs.

6.1.3.4 operator"!=()

```
static bool PhyloTree.Formats.Attribute.operator!= (
            Attribute left,
            Attribute right )  [static]
```

Compares two Attributes (negated).

Parameters

left	The first Attribute to compare.
right	The second Attribute to compare.

Returns

`false` if both Attributes have the same AttributeName (case insensitive) and value for IsNumeric. `true` otherwise.

Definition at line 758 of file Binary.cs.

6.1.3.5 operator==()

```
static bool PhyloTree.Formats.Attribute.operator== (
            Attribute left,
            Attribute right ) [static]
```

Compares two Attributes.

Parameters

left	The first Attribute to compare.
right	The second Attribute to compare.

Returns

`true` if both Attributes have the same AttributeName (case insensitive) and value for IsNumeric. `false` otherwise.

Definition at line 747 of file Binary.cs.

6.1.4 Property Documentation

6.1.4.1 AttributeName

```
string PhyloTree.Formats.Attribute.AttributeName [get]
```

The name of the attribute.

Definition at line 697 of file Binary.cs.

6.1.4.2 IsNumeric

```
bool PhyloTree.Formats.Attribute.IsNumeric  [get]
```

Whether the attribute is represented by a numeric value or a string.

Definition at line 702 of file Binary.cs.

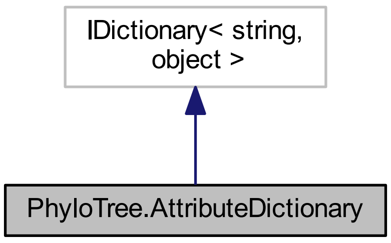
The documentation for this struct was generated from the following file:

- Binary.cs

6.2 PhyloTree.AttributeDictionary Class Reference

Represents the attributes of a node. Attributes Name, Length and Support are always included. See the respective properties for default values.

Inheritance diagram for PhyloTree.AttributeDictionary:



Public Member Functions

- void Add (string name, object value)

  Adds an attribute with the specified name and value to the AttributeDictionary. Throws an exception if the AttributeDictionary already contains an attribute with the same name .
- void Add (KeyValuePair< string, object > item)

  Adds an attribute with the specified name and value to the AttributeDictionary. Throws an exception if the AttributeDictionary already contains an attribute with the same name.
- void Clear ()

  Removes all attributes from the dictionary, except the "`Name`", "`Length`" and "`Support`" attributes.
- bool Contains (KeyValuePair< string, object > item)

  Determines whether the AttributeDictionary contains the specified item .
- bool ContainsKey (string name)

  Determines whether the AttributeDictionary contains an attribute with the specified name name .
- void CopyTo (KeyValuePair< string, object >[ ] array, int arrayIndex)

  Copies the elements of the AttributeDictionary to an array, starting at a specific array index.
- IEnumerator< KeyValuePair< string, object > > GetEnumerator ()

Returns an enumerator that iterates through the AttributeDictionary.

- bool Remove (string name)

  Removes the attribute with the specified name from the AttributeDictionary. Attributes `"Name"`, `"Length"` *and* `"Support"` *cannot be removed.*

- bool Remove (KeyValuePair< string, object > item)

  Removes the attribute with the specified name from the AttributeDictionary. Attributes `"Name"`, `"Length"` *and* `"Support"` *cannot be removed.*

- bool TryGetValue (string name, out object value)

  Gets the value of the attribute with the specified name . Getting the value of attributes `"Name"`, `"Length"` *and* `"Support"` *does not require a dictionary lookup.*

- AttributeDictionary ()

  Constructs an AttributeDictionary containing only the `"Name"`, `"Length"` *and* `"Support"` *attributes.*

Public Attributes

- ICollection< string > Keys => InternalStorage.Keys

  Gets a collection containing the names of the attributes in the AttributeDictionary.

- ICollection< object > Values => InternalStorage.Values

  Gets a collection containing the values of the attributes in the AttributeDictionary.

- int Count => InternalStorage.Count

  Gets the number of attributes contained in the AttributeDictionary.

- bool IsReadOnly => false

  Determine whether the AttributeDictionary is read-only. This is always `false` *in the current implementation.*

Properties

- string Name `[get, set]`

  The name of this node (e.g. the species name for leaf nodes). Default is `""`. *Getting the value of this property does not require a dictionary lookup.*

- double Length `[get, set]`

  The length of the branch leading to this node. This is `double.NaN` *for branches whose length is not specified (e.g. the root node). Getting the value of this property does not require a dictionary lookup.*

- double Support `[get, set]`

  The support value of this node. This is `double.NaN` *for branches whose support is not specified. The interpretation of the support value depends on how the tree was built. Getting the value of this property does not require a dictionary lookup.*

- object this[string name] `[get, set]`

  Gets or sets the value of the attribute with the specified name . Getting the value of attributes `"Name"`, `"Length"` *and* `"Support"` *does not require a dictionary lookup.*

6.2.1 Detailed Description

Represents the attributes of a node. Attributes Name, Length and Support are always included. See the respective properties for default values.

Definition at line 13 of file AttributeDictionary.cs.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 AttributeDictionary()

```
PhyloTree.AttributeDictionary.AttributeDictionary ( )
```

Constructs an AttributeDictionary containing only the `"Name"`, `"Length"` and `"Support"` attributes.

Definition at line 294 of file AttributeDictionary.cs.

6.2.3 Member Function Documentation

6.2.3.1 Add() [1/2]

```
void PhyloTree.AttributeDictionary.Add (
             KeyValuePair< string, object > item )
```

Adds an attribute with the specified name and value to the AttributeDictionary. Throws an exception if the AttributeDictionary already contains an attribute with the same name.

Parameters

item	The item to be added to the dictionary.

Definition at line 153 of file AttributeDictionary.cs.

6.2.3.2 Add() [2/2]

```
void PhyloTree.AttributeDictionary.Add (
             string name,
             object value )
```

Adds an attribute with the specified *name* and *value* to the AttributeDictionary. Throws an exception if the AttributeDictionary already contains an attribute with the same *name* .

Parameters

name	The name of the attribute.
value	The value of the attribute.

Definition at line 144 of file AttributeDictionary.cs.

6.2.3.3 Clear()

```
void PhyloTree.AttributeDictionary.Clear ( )
```

Removes all attributes from the dictionary, except the `"Name"`, `"Length"` and `"Support"` attributes.

Definition at line 161 of file AttributeDictionary.cs.

6.2.3.4 Contains()

```
bool PhyloTree.AttributeDictionary.Contains (
            KeyValuePair< string, object > item )
```

Determines whether the AttributeDictionary contains the specified *item* .

Parameters

item	The item to locate in the AttributeDictionary

Returns

`true` if the AttributeDictionary contains the specified *item* , `false` otherwise.

Definition at line 179 of file AttributeDictionary.cs.

6.2.3.5 ContainsKey()

```
bool PhyloTree.AttributeDictionary.ContainsKey (
            string name )
```

Determines whether the AttributeDictionary contains an attribute with the specified name *name* .

Parameters

name	The name of the attribute to locate.

Returns

`true` if the AttributeDictionary contains an attribute with the specified *name* , `false` otherwise.

Definition at line 189 of file AttributeDictionary.cs.

6.2.3.6 CopyTo()

```
void PhyloTree.AttributeDictionary.CopyTo (
            KeyValuePair< string, object >[] array,
            int arrayIndex )
```

Copies the elements of the AttributeDictionary to an array, starting at a specific array index.

Parameters

array	The array to which the elements will be copied.
arrayIndex	The index at which to start copying.

Definition at line 199 of file AttributeDictionary.cs.

6.2.3.7 GetEnumerator()

```
IEnumerator<KeyValuePair<string, object> > PhyloTree.AttributeDictionary.GetEnumerator ( )
```

Returns an enumerator that iterates through the AttributeDictionary.

Returns

An enumerator that iterates through the AttributeDictionary.

Definition at line 214 of file AttributeDictionary.cs.

6.2.3.8 Remove() [1/2]

```
bool PhyloTree.AttributeDictionary.Remove (
             KeyValuePair< string, object > item )
```

Removes the attribute with the specified name from the AttributeDictionary. Attributes `"Name"`, `"Length"` and `"Support"` cannot be removed.

Parameters

item	The attribute to remove (only the name will be used).

Returns

A `bool` indicating whether the attribute was succesfully removed.

Definition at line 241 of file AttributeDictionary.cs.

6.2.3.9 Remove() [2/2]

```
bool PhyloTree.AttributeDictionary.Remove (
             string name )
```

Removes the attribute with the specified name from the AttributeDictionary. Attributes `"Name"`, `"Length"` and `"Support"` cannot be removed.

Parameters

name	The name of the attribute to remove.

Returns

A `bool` indicating whether the attribute was succesfully removed.

Definition at line 224 of file AttributeDictionary.cs.

6.2.3.10 TryGetValue()

```
bool PhyloTree.AttributeDictionary.TryGetValue (
             string name,
             out object value )
```

Gets the value of the attribute with the specified *name* . Getting the value of attributes `"Name"`, `"Length"` and `"Support"` does not require a dictionary lookup.

Parameters

name	The name of the attribute to get.
value	When this method returns, contains the value of the attribute with the specified *name* , if this is found in the AttributeDictionary, or `null` otherwise.

Returns

A `bool` indicating whether an attribute with the specified *name* was found in the AttributeDictionary.

Definition at line 259 of file AttributeDictionary.cs.

6.2.4 Member Data Documentation

6.2.4.1 Count

```
int PhyloTree.AttributeDictionary.Count => InternalStorage.Count
```

Gets the number of attributes contained in the AttributeDictionary.

Definition at line 132 of file AttributeDictionary.cs.

6.2.4.2 IsReadOnly

```
bool PhyloTree.AttributeDictionary.IsReadOnly => false
```

Determine whether the AttributeDictionary is read-only. This is always `false` in the current implementation.

Definition at line 137 of file AttributeDictionary.cs.

6.2.4.3 Keys

```
ICollection<string> PhyloTree.AttributeDictionary.Keys => InternalStorage.Keys
```

Gets a collection containing the names of the attributes in the AttributeDictionary.

Definition at line 122 of file AttributeDictionary.cs.

6.2.4.4 Values

```
ICollection<object> PhyloTree.AttributeDictionary.Values => InternalStorage.Values
```

Gets a collection containing the values of the attributes in the AttributeDictionary.

Definition at line 127 of file AttributeDictionary.cs.

6.2.5 Property Documentation

6.2.5.1 Length

```
double PhyloTree.AttributeDictionary.Length  [get], [set]
```

The length of the branch leading to this node. This is `double.NaN` for branches whose length is not specified (e.g. the root node). Getting the value of this property does not require a dictionary lookup.

Definition at line 40 of file AttributeDictionary.cs.

6.2.5.2 Name

```
string PhyloTree.AttributeDictionary.Name  [get], [set]
```

The name of this node (e.g. the species name for leaf nodes). Default is `""`. Getting the value of this property does not require a dictionary lookup.

Definition at line 22 of file AttributeDictionary.cs.

6.2.5.3 Support

```
double PhyloTree.AttributeDictionary.Support  [get], [set]
```

The support value of this node. This is `double.NaN` for branches whose support is not specified. The interpretation of the support value depends on how the tree was built. Getting the value of this property does not require a dictionary lookup.

Definition at line 58 of file AttributeDictionary.cs.

6.2.5.4 this[string name]

```
object PhyloTree.AttributeDictionary.this[string name]  [get], [set]
```

Gets or sets the value of the attribute with the specified *name* . Getting the value of attributes `"Name"`, `"Length"` and `"Support"` does not require a dictionary lookup.

Parameters

name	The name of the attribute to get/set.

Returns

The value of the attribute, boxed into an `object`.

Definition at line 76 of file AttributeDictionary.cs.

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

- AttributeDictionary.cs

6.3 PhyloTree.Formats.BinaryTree Class Reference

Contains methods to read and write tree files in binary format.

Static Public Member Functions

- static bool HasValidTrailer (Stream inputStream, bool keepOpen=false)

  Determines whether the tree file stream has a valid trailer.
- static bool IsValidStream (Stream inputStream, bool keepOpen=false)

  Determines whether the tree file stream is valid (i.e. it has a valid header).
- static BinaryTreeMetadata ParseMetadata (Stream inputStream, bool keepOpen=false, BinaryReader reader=null, Action< double > progressAction=null)

  Reads the metadata from a file containing trees in binary format.
- static IEnumerable< TreeNode > ParseTrees (Stream inputStream, bool keepOpen=false, Action< double > progressAction=null)

Lazily parses trees from a file in binary format. Each tree in the file is not read and parsed until it is requested.

- static List< TreeNode > ParseAllTrees (Stream inputStream, bool keepOpen=false, Action< double > progressAction=null)

  Parses trees from a file in binary format and completely loads them in memory.
- static IEnumerable< TreeNode > ParseTrees (string inputFile, Action< double > progressAction=null)

  Lazily parses trees from a file in binary format. Each tree in the file is not read and parsed until it is requested.
- static List< TreeNode > ParseAllTrees (string inputFile, Action< double > progressAction=null)

  Parses trees from a file in binary format and completely loads them in memory.
- static void WriteTree (TreeNode tree, Stream outputStream, bool keepOpen=false, Stream additionalData↩ToCopy=null)

  Writes a single tree in Binary format.
- static void WriteTree (TreeNode tree, string outputFile, bool append=false, Stream additionalDataTo↩Copy=null)

  Writes a single tree in Binary format.
- static void WriteAllTrees (IEnumerable< TreeNode > trees, string outputFile, bool append=false, Action< int > progressAction=null, Stream additionalDataToCopy=null)

  Writes trees in binary format.
- static void WriteAllTrees (IEnumerable< TreeNode > trees, Stream outputStream, bool keepOpen=false, Action< int > progressAction=null, Stream additionalDataToCopy=null)

  Writes trees in binary format.
- static void WriteAllTrees (IList< TreeNode > trees, string outputFile, bool append=false, Action< double > progressAction=null, Stream additionalDataToCopy=null)

  Writes trees in binary format.
- static void WriteAllTrees (IList< TreeNode > trees, Stream outputStream, bool keepOpen=false, Action< double > progressAction=null, Stream additionalDataToCopy=null)

  Writes trees in binary format.

6.3.1 Detailed Description

Contains methods to read and write tree files in binary format.

Definition at line 16 of file Binary.cs.

6.3.2 Member Function Documentation

6.3.2.1 HasValidTrailer()

```
static bool PhyloTree.Formats.BinaryTree.HasValidTrailer (
            Stream inputStream,
            bool keepOpen = false )  [static]
```

Determines whether the tree file stream has a valid trailer.

Parameters

inputStream	The Stream from which the file should be read. Its Stream.CanSeek must be `true`. It does not have to be a FileStream.
keepOpen	Determines whether the stream should be disposed at the end of this method or not.

Returns

`true` if the *inputStream* has a valid trailer, `false` otherwise.

Definition at line 24 of file Binary.cs.

6.3.2.2 IsValidStream()

```
static bool PhyloTree.Formats.BinaryTree.IsValidStream (
            Stream inputStream,
            bool keepOpen = false )  [static]
```

Determines whether the tree file stream is valid (i.e. it has a valid header).

Parameters

inputStream	The Stream from which the file should be read. Its Stream.CanSeek must be `true`. It does not have to be a FileStream.
keepOpen	Determines whether the stream should be disposed at the end of this method or not.

Returns

`true` if the *inputStream* has a valid header, `false` otherwise.

Definition at line 61 of file Binary.cs.

6.3.2.3 ParseAllTrees() `[1/2]`

```
static List<TreeNode> PhyloTree.Formats.BinaryTree.ParseAllTrees (
            Stream inputStream,
            bool keepOpen = false,
            Action< double > progressAction = null )  [static]
```

Parses trees from a file in binary format and completely loads them in memory.

Parameters

inputStream	The Stream from which the file should be read. Its Stream.CanSeek must be `true`. It does not have to be a FileStream.
keepOpen	Determines whether the stream should be disposed at the end of this method or not.
progressAction	An Action that might be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.

Returns

A List<T> containing the trees defined in the file.

Definition at line 384 of file Binary.cs.

6.3.2.4 ParseAllTrees() [2/2]

```
static List<TreeNode> PhyloTree.Formats.BinaryTree.ParseAllTrees (
            string inputFile,
            Action< double > progressAction = null )  [static]
```

Parses trees from a file in binary format and completely loads them in memory.

Parameters

inputFile	The path to the input file.
progressAction	An Action that might be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.

Returns

A List<T> containing the trees defined in the file.

Definition at line 407 of file Binary.cs.

6.3.2.5 ParseMetadata()

```
static BinaryTreeMetadata PhyloTree.Formats.BinaryTree.ParseMetadata (
            Stream inputStream,
            bool keepOpen = false,
            BinaryReader reader = null,
            Action< double > progressAction = null )  [static]
```

Reads the metadata from a file containing trees in binary format.

Parameters

inputStream	The Stream from which the file should be read. Its Stream.CanSeek must be `true`. It does not have to be a FileStream.
keepOpen	Determines whether the stream should be disposed at the end of this method or not.
reader	A BinaryReader to read from the *inputStream* . If this is `null`, a new BinaryReader will be initialised and disposed within this method.
progressAction	An Action that may be invoked while parsing the tree file, with an argument ranging from 0 to 1 describing the progress made in reading the file (determined by the position in the stream).

Returns

A BinaryTreeMetadata object containing metadata information about the tree file.

Definition at line 108 of file Binary.cs.

6.3.2.6 ParseTrees() [1/2]

```
static IEnumerable<TreeNode> PhyloTree.Formats.BinaryTree.ParseTrees (
            Stream inputStream,
            bool keepOpen = false,
            Action< double > progressAction = null )  [static]
```

Lazily parses trees from a file in binary format. Each tree in the file is not read and parsed until it is requested.

Parameters

inputStream	The Stream from which the file should be read. Its Stream.CanSeek must be `true`. It does not have to be a FileStream.
keepOpen	Determines whether the stream should be disposed at the end of this method or not.
progressAction	An Action that might be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.

Returns

A lazy IEnumerable<T> containing the trees defined in the file.

Definition at line 252 of file Binary.cs.

6.3.2.7 ParseTrees() [2/2]

```
static IEnumerable<TreeNode> PhyloTree.Formats.BinaryTree.ParseTrees (
            string inputFile,
            Action< double > progressAction = null )  [static]
```

Lazily parses trees from a file in binary format. Each tree in the file is not read and parsed until it is requested.

Parameters

inputFile	The path to the input file.
progressAction	An Action that might be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.

Returns

A lazy IEnumerable<T> containing the trees defined in the file.

Definition at line 395 of file Binary.cs.

6.3.2.8 WriteAllTrees() [1/4]

```
static void PhyloTree.Formats.BinaryTree.WriteAllTrees (
              IEnumerable< TreeNode > trees,
              Stream outputStream,
              bool keepOpen = false,
              Action< int > progressAction = null,
              Stream additionalDataToCopy = null )  [static]
```

Writes trees in binary format.

Parameters

trees	An IEnumerable<T> containing the trees to be written. It will ony be enumerated once.
outputStream	The Stream on which the trees should be written.
keepOpen	Determines whether the *outputStream* should be kept open after the end of this method.
progressAction	An Action that will be invoked after each tree is written, with the number of trees written so far.
additionalDataToCopy	A stream containing additional data that will be copied into the binary file.

Definition at line 460 of file Binary.cs.

6.3.2.9 WriteAllTrees() [2/4]

```
static void PhyloTree.Formats.BinaryTree.WriteAllTrees (
              IEnumerable< TreeNode > trees,
              string outputFile,
              bool append = false,
              Action< int > progressAction = null,
              Stream additionalDataToCopy = null )  [static]
```

Writes trees in binary format.

Parameters

trees	An IEnumerable<T> containing the trees to be written. It will ony be enumerated once.
outputFile	The file on which the trees should be written.
append	Specifies whether the file should be overwritten or appended to.
progressAction	An Action that will be invoked after each tree is written, with the number of trees written so far.
additionalDataToCopy	A stream containing additional data that will be copied into the binary file.

Definition at line 446 of file Binary.cs.

6.3.2.10 WriteAllTrees() [3/4]

```
static void PhyloTree.Formats.BinaryTree.WriteAllTrees (
             IList< TreeNode > trees,
             Stream outputStream,
             bool keepOpen = false,
             Action< double > progressAction = null,
             Stream additionalDataToCopy = null ) [static]
```

Writes trees in binary format.

Parameters

trees	A collection of trees to be written. Each tree will be accessed twice.
outputStream	The Stream on which the trees should be written.
keepOpen	Determines whether the *outputStream* should be kept open after the end of this method.
progressAction	An Action that will be invoked after each tree is written, with a value between 0 and 1 depending on how many trees have been written so far.
additionalDataToCopy	A stream containing additional data that will be copied into the binary file.

Definition at line 524 of file Binary.cs.

6.3.2.11 WriteAllTrees() [4/4]

```
static void PhyloTree.Formats.BinaryTree.WriteAllTrees (
             IList< TreeNode > trees,
             string outputFile,
             bool append = false,
             Action< double > progressAction = null,
             Stream additionalDataToCopy = null ) [static]
```

Writes trees in binary format.

Parameters

trees	A collection of trees to be written. Each tree will be accessed twice.
outputFile	The file on which the trees should be written.
append	Specifies whether the file should be overwritten or appended to.
progressAction	An Action that will be invoked after each tree is written, with a value between 0 and 1 depending on how many trees have been written so far.
additionalDataToCopy	A stream containing additional data that will be copied into the binary file.

Definition at line 509 of file Binary.cs.

6.3.2.12 WriteTree() [1/2]

```
static void PhyloTree.Formats.BinaryTree.WriteTree (
            TreeNode tree,
            Stream outputStream,
            bool keepOpen = false,
            Stream additionalDataToCopy = null )  [static]
```

Writes a single tree in Binary format.

Parameters

tree	The tree to be written.
outputStream	The Stream on which the tree should be written.
keepOpen	Determines whether the *outputStream* should be kept open after the end of this method.
additionalDataToCopy	A stream containing additional data that will be copied into the binary file.

Definition at line 420 of file Binary.cs.

6.3.2.13 WriteTree() [2/2]

```
static void PhyloTree.Formats.BinaryTree.WriteTree (
            TreeNode tree,
            string outputFile,
            bool append = false,
            Stream additionalDataToCopy = null )  [static]
```

Writes a single tree in Binary format.

Parameters

tree	The tree to be written.
outputFile	The file on which the trees should be written.
append	Specifies whether the file should be overwritten or appended to.
additionalDataToCopy	A stream containing additional data that will be copied into the binary file.

Definition at line 432 of file Binary.cs.

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

- Binary.cs

6.4 PhyloTree.Formats.BinaryTreeMetadata Class Reference

Holds metadata information about a file containing trees in binary format.

Properties

- IEnumerable< long > TreeAddresses `[get, set]`

  The addresses of the trees (i.e. byte offsets from the start of the file).
- bool GlobalNames `[get, set]`

  Determines whether there are any global names stored in the file's header that are used when parsing the trees.
- IReadOnlyList< string > Names `[get, set]`

  Contains any global names stored in the file's header that are used when parsing the trees.
- IReadOnlyList< Attribute > AllAttributes `[get, set]`

  Contains any global attributes stored in the file's header that are used when parsing the trees.

6.4.1 Detailed Description

Holds metadata information about a file containing trees in binary format.

Definition at line 666 of file Binary.cs.

6.4.2 Property Documentation

6.4.2.1 AllAttributes

```
IReadOnlyList<Attribute> PhyloTree.Formats.BinaryTreeMetadata.AllAttributes  [get], [set]
```

Contains any global attributes stored in the file's header that are used when parsing the trees.

Definition at line 686 of file Binary.cs.

6.4.2.2 GlobalNames

```
bool PhyloTree.Formats.BinaryTreeMetadata.GlobalNames  [get], [set]
```

Determines whether there are any global names stored in the file's header that are used when parsing the trees.

Definition at line 676 of file Binary.cs.

6.4.2.3 Names

```
IReadOnlyList<string> PhyloTree.Formats.BinaryTreeMetadata.Names  [get], [set]
```

Contains any global names stored in the file's header that are used when parsing the trees.

Definition at line 681 of file Binary.cs.

6.4.2.4 TreeAddresses

```
IEnumerable<long> PhyloTree.Formats.BinaryTreeMetadata.TreeAddresses  [get], [set]
```

The addresses of the trees (i.e. byte offsets from the start of the file).

Definition at line 671 of file Binary.cs.

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

- Binary.cs

6.5 PhyloTree.Formats.NEXUS Class Reference

Contains methods to read and write trees in NEXUS format.

Static Public Member Functions

- static List< TreeNode > ParseAllTrees (string sourceString=null, Stream sourceStream=null, bool keep↩ Open=false, Action< double > progressAction=null)

  Parses a NEXUS file and completely loads it into memory. Can be used to parse a string or a file.
- static IEnumerable< TreeNode > ParseTrees (string inputFile, Action< double > progressAction=null)

  Lazily parses a NEXUS file. Each tree in the NEXUS file is not read and parsed until it is requested. Can be used to parse a string or a Stream.
- static IEnumerable< TreeNode > ParseTrees (string sourceString=null, Stream sourceStream=null, bool keepOpen=false, Action< double > progressAction=null)

  Lazily parses a NEXUS file. Each tree in the NEXUS file is not read and parsed until it is requested. Can be used to parse a string or a Stream.
- static IEnumerable< TreeNode > ParseTrees (Stream inputStream, bool keepOpen=false, Action< double > progressAction=null)

  Lazily parses trees from a file in NEXUS format. Each tree in the file is not read and parsed until it is requested.
- static List< TreeNode > ParseAllTrees (string inputFile, Action< double > progressAction=null)

  Parses trees from a file in NEXUS format and completely loads them in memory.
- static List< TreeNode > ParseAllTrees (Stream inputStream, bool keepOpen=false, Action< double > progressAction=null)

  Parses trees from a file in NEXUS format and completely loads them in memory.
- static void WriteTree (TreeNode tree, Stream outputStream, bool keepOpen=false, bool translate=true, bool translateQuotes=true, TextReader additionalNexusBlocks=null)

  Writes a single tree in NEXUS format.
- static void WriteTree (TreeNode tree, string outputFile, bool append=false, bool translate=true, bool translateQuotes=true, TextReader additionalNexusBlocks=null)

  Writes a single tree in NEXUS format.
- static void WriteAllTrees (IList< TreeNode > trees, string outputFile, bool append=false, Action< double > progressAction=null, bool translate=true, bool translateQuotes=true, TextReader additionalNexusBlocks=null)

  Writes trees in NEXUS format.
- static void WriteAllTrees (IList< TreeNode > trees, Stream outputStream, bool keepOpen=false, Action< double > progressAction=null, bool translate=true, bool translateQuotes=true, TextReader additionalNexus↩ Blocks=null)

  Writes trees in NEXUS format.
- static void WriteAllTrees (IEnumerable< TreeNode > trees, string outputFile, bool append=false, Action< int > progressAction=null, TextReader additionalNexusBlocks=null)

  Writes trees in NEXUS format.
- static void WriteAllTrees (IEnumerable< TreeNode > trees, Stream outputStream, bool keepOpen=false, Action< int > progressAction=null, TextReader additionalNexusBlocks=null)

  Writes trees in NEXUS format.

6.5.1 Detailed Description

Contains methods to read and write trees in NEXUS format.

Definition at line 14 of file NEXUS.cs.

6.5.2 Member Function Documentation

6.5.2.1 ParseAllTrees() `[1/3]`

```
static List<TreeNode> PhyloTree.Formats.NEXUS.ParseAllTrees (
            Stream inputStream,
            bool keepOpen = false,
            Action< double > progressAction = null )  [static]
```

Parses trees from a file in NEXUS format and completely loads them in memory.

Parameters

inputStream	The Stream from which the file should be read.
keepOpen	Determines whether the stream should be disposed at the end of this method or not.
progressAction	An Action that will be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.

Returns

A List<T> containing the trees defined in the file.

Definition at line 406 of file NEXUS.cs.

6.5.2.2 ParseAllTrees() `[2/3]`

```
static List<TreeNode> PhyloTree.Formats.NEXUS.ParseAllTrees (
            string inputFile,
            Action< double > progressAction = null )  [static]
```

Parses trees from a file in NEXUS format and completely loads them in memory.

Parameters

inputFile	The path to the input file.
progressAction	An Action that will be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.

Returns

A List<T> containing the trees defined in the file.

Definition at line 393 of file NEXUS.cs.

6.5.2.3 ParseAllTrees() [3/3]

```
static List<TreeNode> PhyloTree.Formats.NEXUS.ParseAllTrees (
            string sourceString = null,
            Stream sourceStream = null,
            bool keepOpen = false,
            Action< double > progressAction = null )  [static]
```

Parses a NEXUS file and completely loads it into memory. Can be used to parse a string or a file.

Parameters

sourceString	The NEXUS file content. If this parameter is specified, *sourceStream* is ignored.
sourceStream	The stream to parse.
keepOpen	Determines whether the stream should be disposed at the end of this method or not.
progressAction	An Action that might be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.

Returns

A List<T> containing the trees defined in the "Trees" blocks of the NEXUS file.

Definition at line 80 of file NEXUS.cs.

6.5.2.4 ParseTrees() [1/3]

```
static IEnumerable<TreeNode> PhyloTree.Formats.NEXUS.ParseTrees (
            Stream inputStream,
            bool keepOpen = false,
            Action< double > progressAction = null )  [static]
```

Lazily parses trees from a file in NEXUS format. Each tree in the file is not read and parsed until it is requested.

Parameters

inputStream	The Stream from which the file should be read.
keepOpen	Determines whether the stream should be disposed at the end of this method or not.
progressAction	An Action that will be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.

Returns

A lazy IEnumerable<T> containing the trees defined in the file.

Definition at line 382 of file NEXUS.cs.

6.5.2.5 ParseTrees() [2/3]

```
static IEnumerable<TreeNode> PhyloTree.Formats.NEXUS.ParseTrees (
            string inputFile,
            Action< double > progressAction = null )  [static]
```

Lazily parses a NEXUS file. Each tree in the NEXUS file is not read and parsed until it is requested. Can be used to parse a string or a Stream.

Parameters

inputFile	The path to the input file.
progressAction	An Action that might be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.

Returns

A lazy IEnumerable<T> containing the trees defined in the "Trees" blocks of the NEXUS file.

Definition at line 93 of file NEXUS.cs.

6.5.2.6 ParseTrees() [3/3]

```
static IEnumerable<TreeNode> PhyloTree.Formats.NEXUS.ParseTrees (
            string sourceString = null,
            Stream sourceStream = null,
            bool keepOpen = false,
            Action< double > progressAction = null )  [static]
```

Lazily parses a NEXUS file. Each tree in the NEXUS file is not read and parsed until it is requested. Can be used to parse a string or a Stream.

Parameters

sourceString	The NEXUS file content. If this parameter is specified, *sourceStream* is ignored.
sourceStream	The stream to parse.
keepOpen	Determines whether the stream should be disposed at the end of this method or not.
progressAction	An Action that might be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.

Returns

A lazy IEnumerable<T> containing the trees defined in the "Trees" blocks of the NEXUS file.

Definition at line 107 of file NEXUS.cs.

6.5.2.7 WriteAllTrees() [1/4]

```
static void PhyloTree.Formats.NEXUS.WriteAllTrees (
             IEnumerable< TreeNode > trees,
             Stream outputStream,
             bool keepOpen = false,
             Action< int > progressAction = null,
             TextReader additionalNexusBlocks = null )  [static]
```

Writes trees in NEXUS format.

Parameters

trees	An IEnumerable<T> containing the trees to be written. It will only be enumerated once.
outputStream	The Stream on which the trees should be written.
keepOpen	Determines whether the *outputStream* should be kept open after the end of this method.
progressAction	An Action that will be invoked after each tree is written, with the number of trees written so far.
additionalNexusBlocks	A TextReader that can read additional NEXUS blocks that will be placed at the end of the file.

Definition at line 610 of file NEXUS.cs.

6.5.2.8 WriteAllTrees() [2/4]

```
static void PhyloTree.Formats.NEXUS.WriteAllTrees (
             IEnumerable< TreeNode > trees,
             string outputFile,
             bool append = false,
             Action< int > progressAction = null,
             TextReader additionalNexusBlocks = null )  [static]
```

Writes trees in NEXUS format.

Parameters

trees	An IEnumerable<T> containing the trees to be written. It will only be enumerated once.
outputFile	The file on which the trees should be written.
append	Specifies whether the file should be overwritten or appended to.
progressAction	An Action that will be invoked after each tree is written, with the number of trees written so far.
additionalNexusBlocks	A TextReader that can read additional NEXUS blocks that will be placed at the end of the file.

Definition at line 596 of file NEXUS.cs.

6.5.2.9 WriteAllTrees() [3/4]

```
static void PhyloTree.Formats.NEXUS.WriteAllTrees (
             IList< TreeNode > trees,
             Stream outputStream,
             bool keepOpen = false,
             Action< double > progressAction = null,
             bool translate = true,
             bool translateQuotes = true,
             TextReader additionalNexusBlocks = null )  [static]
```

Writes trees in NEXUS format.

Parameters

trees	A collection of trees to be written. If *translate* is `true`, each tree will be accessed twice. Otherwise, each tree will be accessed once.
outputStream	The Stream on which the trees should be written.
keepOpen	Determines whether the *outputStream* should be kept open after the end of this method.
progressAction	An Action that will be invoked after each tree is written, with a value between 0 and 1 depending on how many trees have been written so far.
translate	If this is `true`, a `Taxa` block and a `Translate` statement in the `Trees` block are added to the NEXUS file.
translateQuotes	If this is `true`, entries in the `Taxa` block and a `Translate` statement in the `Trees` block are placed between single quotes. Otherwise, they are not. This has no effect if *translate* is `false`.
additionalNexusBlocks	A TextReader that can read additional NEXUS blocks that will be placed at the end of the file.

Definition at line 466 of file NEXUS.cs.

6.5.2.10 WriteAllTrees() [4/4]

```
static void PhyloTree.Formats.NEXUS.WriteAllTrees (
             IList< TreeNode > trees,
             string outputFile,
             bool append = false,
             Action< double > progressAction = null,
             bool translate = true,
             bool translateQuotes = true,
             TextReader additionalNexusBlocks = null )  [static]
```

Writes trees in NEXUS format.

Parameters

trees	A collection of trees to be written. If *translate* is `true`, each tree will be accessed twice. Otherwise, each tree will be accessed once.
outputFile	The file on which the trees should be written.
append	Specifies whether the file should be overwritten or appended to.
progressAction	An Action that will be invoked after each tree is written, with a value between 0 and 1 depending on how many trees have been written so far.
translate	If this is `true`, a `Taxa` block and a `Translate` statement in the `Trees` block are added to the NEXUS file.
translateQuotes	If this is `true`, entries in the `Taxa` block and a `Translate` statement in the `Trees` block are placed between single quotes. Otherwise, they are not. This has no effect if *translate* is `false`.
additionalNexusBlocks	A TextReader that can read additional NEXUS blocks that will be placed at the end of the file.

Definition at line 450 of file NEXUS.cs.

6.5.2.11 WriteTree() [1/2]

```
static void PhyloTree.Formats.NEXUS.WriteTree (
            TreeNode tree,
            Stream outputStream,
            bool keepOpen = false,
            bool translate = true,
            bool translateQuotes = true,
            TextReader additionalNexusBlocks = null )  [static]
```

Writes a single tree in NEXUS format.

Parameters

tree	The tree to be written.
outputStream	The Stream on which the tree should be written.
keepOpen	Determines whether the *outputStream* should be kept open after the end of this method.
translate	If this is `true`, a `Taxa` block and a `Translate` statement in the `Trees` block are added to the NEXUS file.
translateQuotes	If this is `true`, entries in the `Taxa` block and a `Translate` statement in the `Trees` block are placed between single quotes. Otherwise, they are not. This has no effect if *translate* is `false`.
additionalNexusBlocks	A TextReader that can read additional NEXUS blocks that will be placed at the end of the file.

Definition at line 420 of file NEXUS.cs.

6.5.2.12 WriteTree() [2/2]

```
static void PhyloTree.Formats.NEXUS.WriteTree (
TreeNode tree,
string outputFile,
bool append = false,
bool translate = true,
bool translateQuotes = true,
TextReader additionalNexusBlocks = null )  [static]
```

Writes a single tree in NEXUS format.

Parameters

tree	The tree to be written.
outputFile	The file on which the tree should be written.
append	Specifies whether the file should be overwritten or appended to.
translate	If this is `true`, a `Taxa` block and a `Translate` statement in the `Trees` block are added to the NEXUS file.
translateQuotes	If this is `true`, entries in the `Taxa` block and a `Translate` statement in the `Trees` block are placed between single quotes. Otherwise, they are not. This has no effect if *translate* is `false`.
additionalNexusBlocks	A TextReader that can read additional NEXUS blocks that will be placed at the end of the file.

Definition at line 434 of file NEXUS.cs.

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

- NEXUS.cs

6.6 PhyloTree.Formats.NWKA Class Reference

Contains methods to read and write trees in Newick and Newick-with-Attributes (NWKA) format.

Static Public Member Functions

- static TreeNode ParseTree (string source, bool debug=false, TreeNode parent=null)

  Parse a Newick-with-Attributes string into a TreeNode object.
- static IEnumerable< TreeNode > ParseTreesFromSource (string source, bool debug=false)

  Lazily parses trees from a string in Newick-with-Attributes (NWKA) format. Each tree in the string is not read and parsed until it is requested.
- static List< TreeNode > ParseAllTreesFromSource (string source, bool debug=false)

  Parses trees from a string in Newick-with-Attributes (NWKA) format and completely loads them in memory.
- static IEnumerable< TreeNode > ParseTrees (string inputFile, Action< double > progressAction=null, bool debug=false)

  Lazily parses trees from a file in Newick-with-Attributes (NWKA) format. Each tree in the file is not read and parsed until it is requested.
- static IEnumerable< TreeNode > ParseTrees (Stream inputStream, bool keepOpen=false, Action< double > progressAction=null, bool debug=false)

  Lazily parses trees from a file in Newick-with-Attributes (NWKA) format. Each tree in the file is not read and parsed until it is requested.
- static List< TreeNode > ParseAllTrees (string inputFile, Action< double > progressAction=null, bool debug=false)

Parses trees from a file in Newick-with-Attributes (NWKA) format and completely loads them in memory.
- static List< TreeNode > ParseAllTrees (Stream inputStream, bool keepOpen=false, Action< double > progressAction=null, bool debug=false)

  Parses trees from a file in Newick-with-Attributes (NWKA) format and completely loads them in memory.
- static string WriteTree (TreeNode tree, bool nwka, bool singleQuoted=false)

  Writes a TreeNode to a string.
- static void WriteTree (TreeNode tree, Stream outputStream, bool keepOpen=false, bool nwka=true, bool singleQuoted=false)

  Writes a single tree in Newick o Newick-with-Attributes format.
- static void WriteTree (TreeNode tree, string outputFile, bool append=false, bool nwka=true, bool single↵ Quoted=false)

  Writes a single tree in Newick o Newick-with-Attributes format.
- static void WriteAllTrees (IEnumerable< TreeNode > trees, string outputFile, bool append=false, Action< int > progressAction=null, bool nwka=true, bool singleQuoted=false)

  Writes trees in Newick o Newick-with-Attributes format.
- static void WriteAllTrees (IEnumerable< TreeNode > trees, Stream outputStream, bool keepOpen=false, Action< int > progressAction=null, bool nwka=true, bool singleQuoted=false)

  Writes trees in Newick o Newick-with-Attributes format.
- static void WriteAllTrees (IList< TreeNode > trees, string outputFile, bool append=false, Action< double > progressAction=null, bool nwka=true, bool singleQuoted=false)

  Writes trees in Newick o Newick-with-Attributes format.
- static void WriteAllTrees (IList< TreeNode > trees, Stream outputStream, bool keepOpen=false, Action< double > progressAction=null, bool nwka=true, bool singleQuoted=false)

  Writes trees in Newick o Newick-with-Attributes format.

6.6.1 Detailed Description

Contains methods to read and write trees in Newick and Newick-with-Attributes (NWKA) format.

Definition at line 15 of file NWKA.cs.

6.6.2 Member Function Documentation

6.6.2.1 ParseAllTrees() `[1/2]`

```
static List<TreeNode> PhyloTree.Formats.NWKA.ParseAllTrees (
             Stream inputStream,
             bool keepOpen = false,
             Action< double > progressAction = null,
             bool debug = false )  [static]
```

Parses trees from a file in Newick-with-Attributes (NWKA) format and completely loads them in memory.

Parameters

inputStream	The Stream from which the file should be read.
keepOpen	Determines whether the stream should be disposed at the end of this method or not.
progressAction	An Action that will be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.
debug	When this is `true`, debug information is printed to the standard output during the parsing.

Returns

A List<T> containing the trees defined in the file.

Definition at line 362 of file NWKA.cs.

6.6.2.2 ParseAllTrees() [2/2]

```
static List<TreeNode> PhyloTree.Formats.NWKA.ParseAllTrees (
            string inputFile,
            Action< double > progressAction = null,
            bool debug = false )  [static]
```

Parses trees from a file in Newick-with-Attributes (NWKA) format and completely loads them in memory.

Parameters

inputFile	The path to the input file.
progressAction	An Action that will be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.
debug	When this is `true`, debug information is printed to the standard output during the parsing.

Returns

A List<T> containing the trees defined in the file.

Definition at line 348 of file NWKA.cs.

6.6.2.3 ParseAllTreesFromSource()

```
static List<TreeNode> PhyloTree.Formats.NWKA.ParseAllTreesFromSource (
            string source,
            bool debug = false )  [static]
```

Parses trees from a string in Newick-with-Attributes (NWKA) format and completely loads them in memory.

Parameters

source	The string from which the trees should be read.
debug	When this is `true`, debug information is printed to the standard output during the parsing.

Returns

A List<T> containing the trees defined in the string.

Definition at line 258 of file NWKA.cs.

6.6.2.4 ParseTree()

```
static TreeNode PhyloTree.Formats.NWKA.ParseTree (
            string source,
            bool debug = false,
            TreeNode parent = null )  [static]
```

Parse a Newick-with-Attributes string into a TreeNode object.

Parameters

source	The Newick-with-Attributes string. This string must specify only a single tree.
parent	The parent node of this node. If parsing a whole tree, this parameter should be left equal to `null`.
debug	When this is `true`, debug information is printed to the standard output during the parsing.

Returns

The parsed TreeNode object.

Definition at line 24 of file NWKA.cs.

6.6.2.5 ParseTrees() [1/2]

```
static IEnumerable<TreeNode> PhyloTree.Formats.NWKA.ParseTrees (
            Stream inputStream,
            bool keepOpen = false,
            Action< double > progressAction = null,
            bool debug = false )  [static]
```

Lazily parses trees from a file in Newick-with-Attributes (NWKA) format. Each tree in the file is not read and parsed until it is requested.

Parameters

inputStream	The Stream from which the file should be read.
keepOpen	Determines whether the stream should be disposed at the end of this method or not.
progressAction	An Action that will be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.
debug	When this is `true`, debug information is printed to the standard output during the parsing.

Returns

A lazy IEnumerable<T> containing the trees defined in the file.

Definition at line 285 of file NWKA.cs.

6.6.2.6 ParseTrees() [2/2]

```
static IEnumerable<TreeNode> PhyloTree.Formats.NWKA.ParseTrees (
            string inputFile,
            Action< double > progressAction = null,
            bool debug = false )  [static]
```

Lazily parses trees from a file in Newick-with-Attributes (NWKA) format. Each tree in the file is not read and parsed until it is requested.

Parameters

inputFile	The path to the input file.
progressAction	An Action that will be called after each tree is parsed, with the approximate progress (as determined by the position in the stream), ranging from 0 to 1.
debug	When this is `true`, debug information is printed to the standard output during the parsing.

Returns

A lazy IEnumerable<T> containing the trees defined in the file.

Definition at line 270 of file NWKA.cs.

6.6.2.7 ParseTreesFromSource()

```
static IEnumerable<TreeNode> PhyloTree.Formats.NWKA.ParseTreesFromSource (
            string source,
            bool debug = false )  [static]
```

Lazily parses trees from a string in Newick-with-Attributes (NWKA) format. Each tree in the string is not read and parsed until it is requested.

Parameters

source	The string from which the trees should be read.
debug	When this is `true`, debug information is printed to the standard output during the parsing.

Returns

A lazy IEnumerable<T> containing the trees defined in the string.

Definition at line 197 of file NWKA.cs.

6.6.2.8 WriteAllTrees() [1/4]

```
static void PhyloTree.Formats.NWKA.WriteAllTrees (
            IEnumerable< TreeNode > trees,
            Stream outputStream,
            bool keepOpen = false,
            Action< int > progressAction = null,
            bool nwka = true,
            bool singleQuoted = false )  [static]
```

Writes trees in Newick o Newick-with-Attributes format.

Parameters

trees	An IEnumerable<T> containing the trees to be written. It will ony be enumerated once.
outputStream	The Stream on which the trees should be written.
keepOpen	Determines whether the *outputStream* should be kept open after the end of this method.
progressAction	An Action that will be invoked after each tree is written, with the number of trees written so far.
nwka	If this is false, a Newick-compliant string is produced for each tree, only including the TreeNode.Name, TreeNode.Length and TreeNode.Support attributes of each branch. Otherwise, a Newick-with-Attributes string is produced, including all attributes.
singleQuoted	If *nwka* is false, this determines whether the names of the nodes are placed between single quotes.

Definition at line 958 of file NWKA.cs.

6.6.2.9 WriteAllTrees() [2/4]

```
static void PhyloTree.Formats.NWKA.WriteAllTrees (
            IEnumerable< TreeNode > trees,
            string outputFile,
            bool append = false,
            Action< int > progressAction = null,
            bool nwka = true,
            bool singleQuoted = false )  [static]
```

Writes trees in Newick o Newick-with-Attributes format.

Parameters

trees	An IEnumerable<T> containing the trees to be written. It will ony be enumerated once.
outputFile	The file on which the trees should be written.
append	Specifies whether the file should be overwritten or appended to.
progressAction	An Action that will be invoked after each tree is written, with the number of trees written so far.
nwka	If this is false, a Newick-compliant string is produced for each tree, only including the TreeNode.Name, TreeNode.Length and TreeNode.Support attributes of each branch. Otherwise, a Newick-with-Attributes string is produced, including all attributes.
singleQuoted	If *nwka* is false, this determines whether the names of the nodes are placed between single quotes.

Definition at line 942 of file NWKA.cs.

6.6.2.10 WriteAllTrees() [3/4]

```
static void PhyloTree.Formats.NWKA.WriteAllTrees (
            IList< TreeNode > trees,
            Stream outputStream,
            bool keepOpen = false,
            Action< double > progressAction = null,
            bool nwka = true,
            bool singleQuoted = false )  [static]
```

Writes trees in Newick o Newick-with-Attributes format.

Parameters

trees	A collection of trees to be written. Each tree will only be accessed once.
outputStream	The Stream on which the trees should be written.
keepOpen	Determines whether the *outputStream* should be kept open after the end of this method.
progressAction	An Action that will be invoked after each tree is written, with a value between 0 and 1 depending on how many trees have been written so far.
nwka	If this is false, a Newick-compliant string is produced for each tree, only including the TreeNode.Name, TreeNode.Length and TreeNode.Support attributes of each branch. Otherwise, a Newick-with-Attributes string is produced, including all attributes.
singleQuoted	If *nwka* is false, this determines whether the names of the nodes are placed between single quotes.

Definition at line 997 of file NWKA.cs.

6.6.2.11 WriteAllTrees() [4/4]

```
static void PhyloTree.Formats.NWKA.WriteAllTrees (
            IList< TreeNode > trees,
            string outputFile,
            bool append = false,
            Action< double > progressAction = null,
            bool nwka = true,
            bool singleQuoted = false )  [static]
```

Writes trees in Newick o Newick-with-Attributes format.

Parameters

trees	A collection of trees to be written. Each tree will only be accessed once.
outputFile	The file on which the trees should be written.
append	Specifies whether the file should be overwritten or appended to.
progressAction	An Action that will be invoked after each tree is written, with a value between 0 and 1 depending on how many trees have been written so far.
nwka	If this is false, a Newick-compliant string is produced for each tree, only including the TreeNode.Name, TreeNode.Length and TreeNode.Support attributes of each branch. Otherwise, a Newick-with-Attributes string is produced, including all attributes.
singleQuoted	If *nwka* is false, this determines whether the names of the nodes are placed between single quotes.

Definition at line 981 of file NWKA.cs.

6.6.2.12 WriteTree() [1/3]

```
static string PhyloTree.Formats.NWKA.WriteTree (
            TreeNode tree,
            bool nwka,
            bool singleQuoted = false )  [static]
```

Writes a TreeNode to a string.

Parameters

tree	The tree to write.
nwka	If this is false, a Newick-compliant string is produced, only including the TreeNode.Name, TreeNode.Length and TreeNode.Support attributes of each branch. Otherwise, a Newick-with-Attributes string is produced, including all attributes.
singleQuoted	If *nwka* is false, this determines whether the names of the nodes are placed between single quotes.

Returns

A string containing the Newick or NWKA representation of the TreeNode.

Definition at line 685 of file NWKA.cs.

6.6.2.13 WriteTree() [2/3]

```
static void PhyloTree.Formats.NWKA.WriteTree (
            TreeNode tree,
            Stream outputStream,
            bool keepOpen = false,
            bool nwka = true,
            bool singleQuoted = false )  [static]
```

Writes a single tree in Newick o Newick-with-Attributes format.

Parameters

tree	The tree to be written.
outputStream	The Stream on which the tree should be written.
keepOpen	Determines whether the *outputStream* should be kept open after the end of this method.
nwka	If this is false, a Newick-compliant string is produced for each tree, only including the TreeNode.Name, TreeNode.Length and TreeNode.Support attributes of each branch. Otherwise, a Newick-with-Attributes string is produced, including all attributes.
singleQuoted	If *nwka* is false, this determines whether the names of the nodes are placed between single quotes.

Definition at line 912 of file NWKA.cs.

6.6.2.14 WriteTree() [3/3]

```
static void PhyloTree.Formats.NWKA.WriteTree (
             TreeNode tree,
             string outputFile,
             bool append = false,
             bool nwka = true,
             bool singleQuoted = false )  [static]
```

Writes a single tree in Newick o Newick-with-Attributes format.

Parameters

tree	The tree to be written.
outputFile	The file on which the tree should be written.
append	Specifies whether the file should be overwritten or appended to.
nwka	If this is false, a Newick-compliant string is produced for each tree, only including the TreeNode.Name, TreeNode.Length and TreeNode.Support attributes of each branch. Otherwise, a Newick-with-Attributes string is produced, including all attributes.
singleQuoted	If *nwka* is false, this determines whether the names of the nodes are placed between single quotes.

Definition at line 926 of file NWKA.cs.

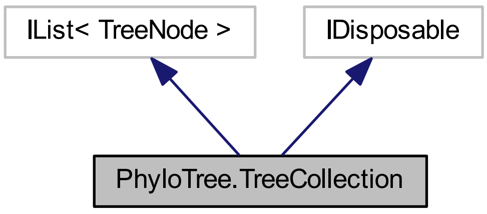
The documentation for this class was generated from the following file:

- NWKA.cs

6.7 PhyloTree.TreeCollection Class Reference

Represents a collection of TreeNode objects. If the full representations of the TreeNode objects reside in memory, this offers the best performance at the expense of memory usage. Alternatively, the trees may be read on demand from a stream in binary format. In this case, accessing any of the trees will require the tree to be parsed. This reduces memory usage, but worsens performance. The internal storage model of the collection is transparent to consumers (except for the difference in performance/memory usage).

Inheritance diagram for PhyloTree.TreeCollection:



Public Member Functions

- void Add (TreeNode item)

  Adds an element to the collection. This is stored in memory, even if the internal storage model of the collection is a Stream.
- void AddRange (IEnumerable< TreeNode > items)

  Adds multiple elements to the collection. These are stored in memory, even if the internal storage model of the collection is a Stream.
- IEnumerator< TreeNode > GetEnumerator ()

  Get an IEnumerator over the collection.
- int IndexOf (TreeNode item)

  Finds the index of the first occurrence of an element in the collection.
- void Insert (int index, TreeNode item)

  Inserts an element in the collection at the specified index.
- void RemoveAt (int index)

  Removes from the collection the element at the specified index.
- void Clear ()

  Removes all elements from the collection. If the internal storage model is a Stream, it is disposed and the internal storage model is converted to a List< TreeNode>.
- bool Contains (TreeNode item)

  Determines whether the collection contains the specified element.
- void CopyTo (TreeNode[ ] array, int arrayIndex)

  Copies the collection to an array.
- bool Remove (TreeNode item)

  Removes the specified element from the collection.
- TreeCollection (List< TreeNode > internalStorage)

  Constructs a TreeCollection object from a List< TreeNode>.
- TreeCollection (Stream binaryTreeStream)

  Constructs a TreeCollection object from a stream of trees in binary format.
- void Dispose ()

  Disposes the TreeCollection, the underlying Stream and StreamReader, and deletes the TemporaryFile (if applicable).

Public Attributes

- bool IsReadOnly => false

  Determine whether the collection is read-only. This is always `false` in the current implementation.

Properties

- Stream UnderlyingStream = null `[get]`

  A stream containing the tree data in binary format, if this is the chosen storage model. This can be either a Memory↩ Stream or a FileStream.
- string TemporaryFile = null `[get, set]`

  If the trees are stored on disk in a temporary file, you should assign this property to the full path of the file. The file will be deleted when the TreeCollection is Dispose()d.
- int Count `[get]`

  The number of trees in the collection.
- TreeNode this[int index] `[get, set]`

  Obtains an element from the collection.

6.7.1 Detailed Description

Represents a collection of TreeNode objects. If the full representations of the TreeNode objects reside in memory, this offers the best performance at the expense of memory usage. Alternatively, the trees may be read on demand from a stream in binary format. In this case, accessing any of the trees will require the tree to be parsed. This reduces memory usage, but worsens performance. The internal storage model of the collection is transparent to consumers (except for the difference in performance/memory usage).

Definition at line 16 of file TreeCollection.cs.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 TreeCollection() `[1/2]`

```
PhyloTree.TreeCollection.TreeCollection (
            List< TreeNode > internalStorage )
```

Constructs a TreeCollection object from a List<TreeNode>.

Parameters

internalStorage	The List<TreeNode> containing the trees to store in the collection. Note that this list is not copied, but used as-is.

Definition at line 429 of file TreeCollection.cs.

6.7.2.2 TreeCollection() `[2/2]`

```
PhyloTree.TreeCollection.TreeCollection (
            Stream binaryTreeStream )
```

Constructs a TreeCollection object from a stream of trees in binary format.

Parameters

binaryTreeStream	The stream of trees in binary format to use. The stream will be disposed when the TreeCollection is disposed. It should not be disposed earlier by external code.

Definition at line 439 of file TreeCollection.cs.

6.7.3 Member Function Documentation

6.7.3.1 Add()

```
void PhyloTree.TreeCollection.Add (
             TreeNode item )
```

Adds an element to the collection. This is stored in memory, even if the internal storage model of the collection is a Stream.

Parameters

item	The element to add.

Definition at line 161 of file TreeCollection.cs.

6.7.3.2 AddRange()

```
void PhyloTree.TreeCollection.AddRange (
             IEnumerable< TreeNode > items )
```

Adds multiple elements to the collection. These are stored in memory, even if the internal storage model of the collection is a Stream.

Parameters

items	The elements to add.

Definition at line 174 of file TreeCollection.cs.

6.7.3.3 Clear()

```
void PhyloTree.TreeCollection.Clear ( )
```

Removes all elements from the collection. If the internal storage model is a Stream, it is disposed and the internal storage model is converted to a List<TreeNode>.

Definition at line 329 of file TreeCollection.cs.

6.7.3.4 Contains()

```
bool PhyloTree.TreeCollection.Contains (
            TreeNode item )
```

Determines whether the collection contains the specified element.

Parameters

item	The element to search for.

Returns

`true` if the collection contains the specified element, `false` otherwise.

Definition at line 356 of file TreeCollection.cs.

6.7.3.5 CopyTo()

```
void PhyloTree.TreeCollection.CopyTo (
            TreeNode[ ] array,
            int arrayIndex )
```

Copies the collection to an array.

Parameters

array	The array in which to copy the collection.
arrayIndex	The index at which to start the copy.

Definition at line 383 of file TreeCollection.cs.

6.7.3.6 Dispose()

```
void PhyloTree.TreeCollection.Dispose ( )
```

Disposes the TreeCollection, the underlying Stream and StreamReader, and deletes the TemporaryFile (if applicable).

Definition at line 513 of file TreeCollection.cs.

6.7.3.7 GetEnumerator()

```
IEnumerator<TreeNode> PhyloTree.TreeCollection.GetEnumerator ( )
```

Get an IEnumerator over the collection.

Returns

An IEnumerator over the collection.

Definition at line 191 of file TreeCollection.cs.

6.7.3.8 IndexOf()

```
int PhyloTree.TreeCollection.IndexOf (
             TreeNode item )
```

Finds the index of the first occurrence of an element in the collection.

Parameters

item	The item to search for.

Returns

The index of the item in the collection.

Definition at line 244 of file TreeCollection.cs.

6.7.3.9 Insert()

```
void PhyloTree.TreeCollection.Insert (
             int index,
             TreeNode item )
```

Inserts an element in the collection at the specified index.

Parameters

index	The index at which to insert the element.
item	The element to insert.

Definition at line 271 of file TreeCollection.cs.

6.7.3.10 Remove()

```
bool PhyloTree.TreeCollection.Remove (
            TreeNode item )
```

Removes the specified element from the collection.

Parameters

item	The element to remove.

Returns

`true` if the removal was successful (i.e. the list contained the element in the first place), `false` otherwise.

Definition at line 404 of file TreeCollection.cs.

6.7.3.11 RemoveAt()

```
void PhyloTree.TreeCollection.RemoveAt (
            int index )
```

Removes from the collection the element at the specified index.

Parameters

index	

Definition at line 298 of file TreeCollection.cs.

6.7.4 Member Data Documentation

6.7.4.1 IsReadOnly

```
bool PhyloTree.TreeCollection.IsReadOnly => false
```

Determine whether the collection is read-only. This is always `false` in the current implementation.

Definition at line 105 of file TreeCollection.cs.

6.7.5 Property Documentation

6.7.5.1 Count

```
int PhyloTree.TreeCollection.Count  [get]
```

The number of trees in the collection.

Definition at line 87 of file TreeCollection.cs.

6.7.5.2 TemporaryFile

```
string PhyloTree.TreeCollection.TemporaryFile = null  [get], [set]
```

If the trees are stored on disk in a temporary file, you should assign this property to the full path of the file. The file will be deleted when the TreeCollection is Dispose()d.

Definition at line 82 of file TreeCollection.cs.

6.7.5.3 this[int index]

```
TreeNode PhyloTree.TreeCollection.this[int index]  [get], [set]
```

Obtains an element from the collection.

Parameters

index	The index of the element to extract.

Returns

The requested element from the collection.

Definition at line 112 of file TreeCollection.cs.

6.7.5.4 UnderlyingStream

```
Stream PhyloTree.TreeCollection.UnderlyingStream = null  [get]
```

A stream containing the tree data in binary format, if this is the chosen storage model. This can be either a MemoryStream or a FileStream.

Definition at line 26 of file TreeCollection.cs.

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

- TreeCollection.cs

6.8 PhyloTree.TreeNode Class Reference

Represents a node in a tree (or a whole tree).

Public Types

- enum NodeRelationship { NodeRelationship.Unknown, NodeRelationship.Ancestor, NodeRelationship.Descendant, NodeRelationship.Relatives }

  Describes the relationship between two nodes.

Public Member Functions

- TreeNode (TreeNode parent)

  Creates a new TreeNode object.
- bool IsRooted ()

  Checks whether the node belongs to a rooted tree.
- TreeNode GetUnrootedTree ()

  Get an unrooted version of the tree.
- TreeNode GetRootedTree (TreeNode outgroup, double position=0.5)

  Get a version of the tree that is rooted at the specified point of the branch leading to the outgroup .
- TreeNode Clone ()

  Recursively clone a TreeNode object.
- List< TreeNode > GetChildrenRecursive ()

  Recursively get all the nodes that descend from this node.
- IEnumerable< TreeNode > GetChildrenRecursiveLazy ()

  Lazily recursively get all the nodes that descend from this node.
- List< TreeNode > GetLeaves ()

  Get all the leaves that descend (directly or indirectly) from this node.
- List< string > GetLeafNames ()

  Get the names of all the leaves that descend (directly or indirectly) from this node.
- List< string > GetNodeNames ()

  Get the names of all the named nodes that descend (directly or indirectly) from this node.
- TreeNode GetNodeFromName (string nodeName)

  Get the child node with the specified name.
- TreeNode GetNodeFromId (string nodeId)

  Get the child node with the specified Id.
- double UpstreamLength ()

  Get the sum of the branch lengths from this node up to the root.
- double LongestDownstreamLength ()

  Get the sum of the branch lengths from this node down to the leaves of the tree. If the tree is not clock-like, the length of the longest path is returned.
- double ShortestDownstreamLength ()

  Get the sum of the branch lengths from this node down to the leaves of the tree. If the tree is not clock-like, the length of the shortest path is returned.
- TreeNode GetRootNode ()

  Get the node of the tree from which all other nodes descend.
- double PathLengthTo (TreeNode otherNode, NodeRelationship nodeRelationship=NodeRelationship.Unknown)

  Get the sum of the branch lengths from this node to the specified node.
- double TotalLength ()

Get the sum of the branch lengths of this node and all its descendants.

- void SortNodes (bool descending)

  Sort (in place) the nodes in the tree in an aesthetically pleasing way.
- override string ToString ()

  Convert the tree to a Newick string.
- bool IsClockLike (double tolerance=0.001)

  Determines whether the tree is clock-like (i.e. all tips are contemporaneous) or not.
- TreeNode GetLastCommonAncestor (params string[ ] monophyleticConstraint)

  Gets the last common ancestor of all the nodes with the specified names, or `null` if the tree doesn't contain all the named nodes.
- TreeNode GetLastCommonAncestor (IEnumerable< string > monophyleticConstraint)

  Gets the last common ancestor of all the nodes with the specified names, or `null` if the tree doesn't contain all the named nodes.
- bool IsLastCommonAncestor (IEnumerable< string > monophyleticConstraint)

  Checks whether this node is the last common ancestor of all the nodes with the specified names.

Static Public Member Functions

- static TreeNode GetLastCommonAncestor (IEnumerable< TreeNode > monophyleticConstraint)

  Gets the last common ancestor of all the specified nodes, or `null` if the tree doesn't contain all the nodes.

Properties

- TreeNode Parent `[get, set]`

  The parent node of this node. This will be `null` for the root node.
- List< TreeNode > Children `[get]`

  The child nodes of this node. This will be empty (but initialised) for leaf nodes.
- AttributeDictionary Attributes = new AttributeDictionary() `[get]`

  The attributes of this node. Attributes Name, Length and Support are always included. See the respective properties for default values.
- double Length `[get, set]`

  The length of the branch leading to this node. This is `double.NaN` for branches whose length is not specified (e.g. the root node).
- double Support `[get, set]`

  The support value of this node. This is `double.NaN` for branches whose support is not specified. The interpretation of the support value depends on how the tree was built.
- string Name `[get, set]`

  The name of this node (e.g. the species name for leaf nodes). Default is `""`.
- string Id `[get]`

  A univocal identifier for the node.

6.8.1 Detailed Description

Represents a node in a tree (or a whole tree).

Definition at line 305 of file TreeNode.cs.

6.8.2 Member Enumeration Documentation

6.8.2.1 NodeRelationship

```
enum PhyloTree.TreeNode.NodeRelationship  [strong]
```

Describes the relationship between two nodes.

Enumerator

Unknown	The relationship between the nodes is unknown.
Ancestor	The first node is an ancestor of the second node.
Descendant	The first node is a descendant of the second node.
Relatives	The two nodes are relatives (i.e. they share a common ancestor which is neither one of them).

Definition at line 806 of file TreeNode.cs.

6.8.3 Constructor & Destructor Documentation

6.8.3.1 TreeNode()

```
PhyloTree.TreeNode.TreeNode (
            TreeNode parent )
```

Creates a new TreeNode object.

Parameters

parent	The parent node of this node. For the root node, this should be `null`.

Definition at line 377 of file TreeNode.cs.

6.8.4 Member Function Documentation

6.8.4.1 Clone()

```
TreeNode PhyloTree.TreeNode.Clone ( )
```

Recursively clone a TreeNode object.

Returns

The cloned TreeNode

Definition at line 553 of file TreeNode.cs.

6.8.4.2 GetChildrenRecursive()

```
List<TreeNode> PhyloTree.TreeNode.GetChildrenRecursive ( )
```

Recursively get all the nodes that descend from this node.

Returns

A List<T> of TreeNode objects, containing the nodes that descend from this node.

Definition at line 583 of file TreeNode.cs.

6.8.4.3 GetChildrenRecursiveLazy()

```
IEnumerable<TreeNode> PhyloTree.TreeNode.GetChildrenRecursiveLazy ( )
```

Lazily recursively get all the nodes that descend from this node.

Returns

An IEnumerable<T> of TreeNode objects, containing the nodes that descend from this node.

Definition at line 601 of file TreeNode.cs.

6.8.4.4 GetLastCommonAncestor() [1/3]

```
TreeNode PhyloTree.TreeNode.GetLastCommonAncestor (
            IEnumerable< string > monophyleticConstraint )
```

Gets the last common ancestor of all the nodes with the specified names, or `null` if the tree doesn't contain all the named nodes.

Parameters

monophyleticConstraint	The collection of names representing nodes whose last common ancestor is to be determined.

Returns

The last common ancestor of all the nodes with the specified names, or `null` if the tree doesn't contain all the named nodes.

Definition at line 1055 of file TreeNode.cs.

6.8.4.5 GetLastCommonAncestor() [2/3]

```
static TreeNode PhyloTree.TreeNode.GetLastCommonAncestor (
            IEnumerable< TreeNode > monophyleticConstraint )  [static]
```

Gets the last common ancestor of all the specified nodes, or `null` if the tree doesn't contain all the nodes.

Parameters

monophyleticConstraint	The collection of nodes whose last common ancestor is to be determined.

Returns

The last common ancestor of all the specified nodes, or `null` if the tree doesn't contain all the nodes.

Definition at line 1021 of file TreeNode.cs.

6.8.4.6 GetLastCommonAncestor() [3/3]

```
TreeNode PhyloTree.TreeNode.GetLastCommonAncestor (
            params string[] monophyleticConstraint )
```

Gets the last common ancestor of all the nodes with the specified names, or `null` if the tree doesn't contain all the named nodes.

Parameters

monophyleticConstraint	The collection of names representing nodes whose last common ancestor is to be determined.

Returns

The last common ancestor of all the nodes with the specified names, or `null` if the tree doesn't contain all the named nodes.

Definition at line 1045 of file TreeNode.cs.

6.8.4.7 GetLeafNames()

```
List<string> PhyloTree.TreeNode.GetLeafNames ( )
```

Get the names of all the leaves that descend (directly or indirectly) from this node.

Returns

A List<T> of strings, containing the names of the leaves that descend from this node.

Definition at line 638 of file TreeNode.cs.

6.8.4.8 GetLeaves()

```
List<TreeNode> PhyloTree.TreeNode.GetLeaves ( )
```

Get all the leaves that descend (directly or indirectly) from this node.

Returns

A List<T> of TreeNode objects, containing the leaves that descend from this node.

Definition at line 618 of file TreeNode.cs.

6.8.4.9 GetNodeFromId()

```
TreeNode PhyloTree.TreeNode.GetNodeFromId (
             string nodeId )
```

Get the child node with the specified Id.

Parameters

node↵ Id	The Id of the node to search.

Returns

The TreeNode object with the specified Id, or `null` if no node with such Id exists.

Definition at line 703 of file TreeNode.cs.

6.8.4.10 GetNodeFromName()

```
TreeNode PhyloTree.TreeNode.GetNodeFromName (
             string nodeName )
```

Get the child node with the specified name.

Parameters

nodeName	The name of the node to search.

Returns

The TreeNode object with the specified name, or `null` if no node with such name exists.

Definition at line 679 of file TreeNode.cs.

6.8.4.11 GetNodeNames()

```
List<string> PhyloTree.TreeNode.GetNodeNames ( )
```

Get the names of all the named nodes that descend (directly or indirectly) from this node.

Returns

A List<T> of strings, containing the names of the named nodes that descend from this node.

Definition at line 658 of file TreeNode.cs.

6.8.4.12 GetRootedTree()

```
TreeNode PhyloTree.TreeNode.GetRootedTree (
             TreeNode outgroup,
             double position = 0.5 )
```

Get a version of the tree that is rooted at the specified point of the branch leading to the *outgroup* .

Parameters

outgroup	The outgroup to be used when rooting the tree.
position	The (relative) position on the branch connecting the outgroup to the rest of the tree on which to place the root.

Returns

A TreeNode containing the rooted tree.

Definition at line 444 of file TreeNode.cs.

6.8.4.13 GetRootNode()

```
TreeNode PhyloTree.TreeNode.GetRootNode ( )
```

Get the node of the tree from which all other nodes descend.

Returns

The node of the tree from which all other nodes descend

Definition at line 793 of file TreeNode.cs.

6.8.4.14 GetUnrootedTree()

```
TreeNode PhyloTree.TreeNode.GetUnrootedTree ( )
```

Get an unrooted version of the tree.

Returns

A TreeNode containing the unrooted tree, having at least 3 children.

Definition at line 397 of file TreeNode.cs.

6.8.4.15 IsClockLike()

```
bool PhyloTree.TreeNode.IsClockLike (
            double tolerance = 0.001 )
```

Determines whether the tree is clock-like (i.e. all tips are contemporaneous) or not.

Parameters

tolerance	The (relative) tolerance when comparing branch lengths.

Returns

A boolean value determining whether the tree is clock-like or not

Definition at line 999 of file TreeNode.cs.

6.8.4.16 IsLastCommonAncestor()

```
bool PhyloTree.TreeNode.IsLastCommonAncestor (
            IEnumerable< string > monophyleticConstraint )
```

Checks whether this node is the last common ancestor of all the nodes with the specified names.

Parameters

monophyleticConstraint	The collection of names representing nodes whose last common ancestor is to be determined.

Returns

`true` if this node is the last common ancestor of all the nodes with the specified names, `false` otherwise.

Definition at line 1079 of file TreeNode.cs.

6.8.4.17 IsRooted()

```
bool PhyloTree.TreeNode.IsRooted ( )
```

Checks whether the node belongs to a rooted tree.

Returns

`true` if the node belongs to a rooted tree, `false` otherwise.

Definition at line 388 of file TreeNode.cs.

6.8.4.18 LongestDownstreamLength()

```
double PhyloTree.TreeNode.LongestDownstreamLength ( )
```

Get the sum of the branch lengths from this node down to the leaves of the tree. If the tree is not clock-like, the length of the longest path is returned.

Returns

The sum of the branch lengths from this node down to the leaves of the tree. If the tree is not clock-like, the length of the longest path is returned.

Definition at line 745 of file TreeNode.cs.

6.8.4.19 PathLengthTo()

```
double PhyloTree.TreeNode.PathLengthTo (
             TreeNode otherNode,
             NodeRelationship nodeRelationship = NodeRelationship.Unknown )
```

Get the sum of the branch lengths from this node to the specified node.

Parameters

otherNode	The node that should be reached
nodeRelationship	A value indicating how this node is related to *otherNode* .

Returns

The sum of the branch lengths from this node to the specified node.

Definition at line 835 of file TreeNode.cs.

6.8.4.20 ShortestDownstreamLength()

```
double PhyloTree.TreeNode.ShortestDownstreamLength ( )
```

Get the sum of the branch lengths from this node down to the leaves of the tree. If the tree is not clock-like, the length of the shortest path is returned.

Returns

The sum of the branch lengths from this node down to the leaves of the tree. If the tree is not clock-like, the length of the shortest path is returned.

Definition at line 769 of file TreeNode.cs.

6.8.4.21 SortNodes()

```
void PhyloTree.TreeNode.SortNodes (
             bool descending )
```

Sort (in place) the nodes in the tree in an aesthetically pleasing way.

Parameters

descending	The way the nodes should be sorted.

Definition at line 920 of file TreeNode.cs.

6.8.4.22 ToString()

```
override string PhyloTree.TreeNode.ToString ( )
```

Convert the tree to a Newick string.

Returns

Definition at line 989 of file TreeNode.cs.

6.8.4.23 TotalLength()

```
double PhyloTree.TreeNode.TotalLength ( )
```

Get the sum of the branch lengths of this node and all its descendants.

Returns

The sum of the branch lengths of this node and all its descendants.

Definition at line 905 of file TreeNode.cs.

6.8.4.24 UpstreamLength()

```
double PhyloTree.TreeNode.UpstreamLength ( )
```

Get the sum of the branch lengths from this node up to the root.

Returns

The sum of the branch lengths from this node up to the root.

Definition at line 726 of file TreeNode.cs.

6.8.5 Property Documentation

6.8.5.1 Attributes

```
AttributeDictionary PhyloTree.TreeNode.Attributes = new AttributeDictionary()  [get]
```

The attributes of this node. Attributes Name, Length and Support are always included. See the respective properties for default values.

Definition at line 320 of file TreeNode.cs.

6.8.5.2 Children

```
List<TreeNode> PhyloTree.TreeNode.Children  [get]
```

The child nodes of this node. This will be empty (but initialised) for leaf nodes.

Definition at line 315 of file TreeNode.cs.

6.8.5.3 Id

```
string PhyloTree.TreeNode.Id  [get]
```

A univocal identifier for the node.

Definition at line 371 of file TreeNode.cs.

6.8.5.4 Length

```
double PhyloTree.TreeNode.Length  [get], [set]
```

The length of the branch leading to this node. This is `double.NaN` for branches whose length is not specified (e.g. the root node).

Definition at line 325 of file TreeNode.cs.

6.8.5.5 Name

```
string PhyloTree.TreeNode.Name  [get], [set]
```

The name of this node (e.g. the species name for leaf nodes). Default is `""`.

Definition at line 355 of file TreeNode.cs.

6.8.5.6 Parent

```
TreeNode PhyloTree.TreeNode.Parent  [get], [set]
```

The parent node of this node. This will be `null` for the root node.

Definition at line 310 of file TreeNode.cs.

6.8.5.7 Support

```
double PhyloTree.TreeNode.Support  [get], [set]
```

The support value of this node. This is `double.NaN` for branches whose support is not specified. The interpretation of the support value depends on how the tree was built.

Definition at line 340 of file TreeNode.cs.

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

- TreeNode.cs

6.9 PhyloTree.Extensions.TypeExtensions Class Reference

Useful extension methods

Static Public Member Functions

- static bool ContainsAll< T > (this IEnumerable< T > haystack, IEnumerable< T > needle)

  Determines whether haystack contains all of the elements in needle .
- static double Median (this IEnumerable< double > array)

  Compute the median of a list of values.
- static bool ContainsAny< T > (this IEnumerable< T > haystack, IEnumerable< T > needle)

  Determines whether haystack contains at least one of the elements in needle .
- static IEnumerable< T > Intersection< T > (this IEnumerable< T > set1, IEnumerable< T > set2)

  Computes the intersection between two sets.
- static TreeNode GetConsensus (this IEnumerable< TreeNode > trees, bool rooted, bool clockLike, double threshold, bool useMedian)

  Constructs a consensus tree.
- static char NextToken (this TextReader reader, ref bool escaping, out bool escaped, ref bool openQuotes, ref bool openApostrophe, out bool eof)

  Reads the next non-whitespace character, taking into account quoting and escaping.
- static string NextWord (this TextReader reader, out bool eof)

  Reads the next word, taking into account whitespaces, square brackets, commas and semicolons.

6.9.1 Detailed Description

Useful extension methods

Definition at line 16 of file Extensions.cs.

6.9.2 Member Function Documentation

6.9.2.1 ContainsAll< T >()

```
static bool PhyloTree.Extensions.TypeExtensions.ContainsAll< T > (
            this IEnumerable< T > haystack,
            IEnumerable< T > needle )  [static]
```

Determines whether *haystack* contains all of the elements in *needle* .

Template Parameters

T	The type of the elements in the collections.

Parameters

haystack	The collection in which to search.
needle	The items to be searched.

Returns

`true` if haystack contains all of the elements that are in needle or needle is empty.

Definition at line 25 of file Extensions.cs.

6.9.2.2 ContainsAny< T >()

```
static bool PhyloTree.Extensions.TypeExtensions.ContainsAny< T > (
             this IEnumerable< T > haystack,
             IEnumerable< T > needle )  [static]
```

Determines whether *haystack* contains at least one of the elements in *needle* .

Template Parameters

T	The type of the elements in the collections.

Parameters

haystack	The collection in which to search.
needle	The items to be searched.

Returns

`true` if haystack contains at least one of the elements that are in needle. Returns `false` if needle is empty.

Definition at line 66 of file Extensions.cs.

6.9.2.3 GetConsensus()

```
static TreeNode PhyloTree.Extensions.TypeExtensions.GetConsensus (
             this IEnumerable< TreeNode > trees,
             bool rooted,
             bool clockLike,
             double threshold,
             bool useMedian )  [static]
```

Constructs a consensus tree.

Parameters

trees	The collection of trees whose consensus is to be computed.
rooted	Whether the consensus tree should be rooted or not.
clockLike	Whether the trees are to be treated as clock-like trees or not. This has an effect on how the branch lengths of the consensus tree are computed.
threshold	The (inclusive) threshold for splits to be included in the consensus tree. Use `0` to get all compatible splits, `0.5` for a majority-rule consensus or `1` for a strict consensus.
useMedian	If this is `true`, the lengths of the branches in the tree will be computed based on the median length/age of the splits used to build the tree. Otherwise, the mean will be used.

Returns

A rooted consensus tree.

Definition at line 110 of file Extensions.cs.

6.9.2.4 Intersection< T >()

```
static IEnumerable<T> PhyloTree.Extensions.TypeExtensions.Intersection< T > (
            this IEnumerable< T > set1,
            IEnumerable< T > set2 )  [static]
```

Computes the intersection between two sets.

Template Parameters

T	The type of the elements in the collections.

Parameters

set1	The first set.
set2	The second set.

Returns

The intersection between the two sets.

Definition at line 87 of file Extensions.cs.

6.9.2.5 Median()

```
static double PhyloTree.Extensions.TypeExtensions.Median (
            this IEnumerable< double > array )  [static]
```

Compute the median of a list of values.

Parameters

array	The list of values whose median is to be computed.

Returns

The median of the list of values.

Definition at line 44 of file Extensions.cs.

6.9.2.6 NextToken()

```
static char PhyloTree.Extensions.TypeExtensions.NextToken (
             this TextReader reader,
             ref bool escaping,
             out bool escaped,
             ref bool openQuotes,
             ref bool openApostrophe,
             out bool eof )  [static]
```

Reads the next non-whitespace character, taking into account quoting and escaping.

Parameters

reader	The TextReader to read from.
escaping	A bool indicating whether the next character will be escaped.
escaped	A bool indicating whether the current character will be escaped.
openQuotes	A bool indicating whether double quotes have been opened.
openApostrophe	A bool indicating whether single quotes have been opened.
eof	A bool indicating whether we have arrived at the end of the file.

Returns

The next non-whitespace character.

Definition at line 164 of file Extensions.cs.

6.9.2.7 NextWord()

```
static string PhyloTree.Extensions.TypeExtensions.NextWord (
             this TextReader reader,
             out bool eof )  [static]
```

Reads the next word, taking into account whitespaces, square brackets, commas and semicolons.

Parameters

reader	The TextReader to read from.
eof	A bool indicating whether we have arrived at the end of the file.

Returns

The next word.

Definition at line 252 of file Extensions.cs.

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

- Extensions.cs

Index