

BinTreeVisualization

Generated by Doxygen 1.10.0

1 General assignment	1
1.1 Roadmap	1
1.2 AVL tree in general	2
1.3 WPF specifics	2
1.4 Layouting nodes	2
1.5 Performance	2
2 Project overview	3
2.1 Base BST	3
2.1.1 Insertion	3
2.1.2 Search	3
2.1.3 Deletion	3
2.2 AVL	4
2.3 App structure	4
3 Binary Tree Visualization	5
3.1 Operations supported	5
3.2 Quality of Lifes	5
3.3 Build	5
4 Namespace Index	7
4.1 Package List	7
5 Hierarchical Index	9
5.1 Class Hierarchy	9
6 Class Index	11
6.1 Class List	11
7 Namespace Documentation	13
7.1 BinTreeVisualization Namespace Reference	13
7.2 BinTreeVisualization.Algorithms Namespace Reference	13
7.2.1 Function Documentation	14
7.2.1.1 BinTreeRow< T >()	14
7.3 BinTreeVisualization.Stats Namespace Reference	14
7.3.1 Function Documentation	15
7.3.1.1 OperationStats()	15
7.4 BinTreeVisualization.UI Namespace Reference	15
8 Class Documentation	17
8.1 BinTreeVisualization.App Class Reference	17
8.1.1 Detailed Description	17
8.2 BinTreeVisualization.Algorithms.BinTree< T > Class Template Reference	17
8.2.1 Detailed Description	20
8.2.2 Member Function Documentation	21

8.2.2.1 AnimAdoption()	21
8.2.2.2 BalanceTreeIfNeeded()	21
8.2.2.3 Blink()	21
8.2.2.4 CreateRoot()	22
8.2.2.5 Delay()	22
8.2.2.6 Delete() [1/2]	22
8.2.2.7 Delete() [2/2]	23
8.2.2.8 Find() [1/3]	23
8.2.2.9 Find() [2/3]	23
8.2.2.10 Find() [3/3]	24
8.2.2.11 FinishOperationStats()	24
8.2.2.12 GetCanvas()	24
8.2.2.13 GetHeight()	24
8.2.2.14 GetMax() [1/2]	25
8.2.2.15 GetMax() [2/2]	25
8.2.2.16 GetMin() [1/2]	25
8.2.2.17 GetMin() [2/2]	25
8.2.2.18 GetRow()	26
8.2.2.19 GetRows()	26
8.2.2.20 Insert() [1/2]	26
8.2.2.21 Insert() [2/2]	26
8.2.2.22 OperationGuard()	27
8.2.2.23 Rotate()	27
8.2.2.24 RotateLeft()	27
8.2.2.25 RotateRight()	28
8.2.2.26 SwapNodes()	28
8.2.2.27 SwapValues()	28
8.2.2.28 VerifyTreeIsNotEmpty()	29
8.2.2.29 VerifyTreeLayout()	29
8.3 BinTreeVisualization.UI.BinTreeControl Class Reference	29
8.3.1 Detailed Description	30
8.3.2 Member Function Documentation	30
8.3.2.1 GetNewScale< T >()	30
8.3.2.2 GetNodesTotalWidth< T >()	31
8.3.2.3 LayoutTreeNSquare< T >()	31
8.3.2.4 ResetText()	32
8.3.2.5 SetScaleAnim()	32
8.3.2.6 SetText()	32
8.3.2.7 VerifyScale< T >()	33
8.4 BinTreeVisualization.UI.BinTreeMan Class Reference	33
8.4.1 Detailed Description	34
8.4.2 Member Function Documentation	34

8.4.2.1 AddRandomItems()	34
8.4.2.2 GetDoubleArg()	35
8.4.2.3 OnDelete()	35
8.4.2.4 OnFind()	35
8.4.2.5 OnGetMax()	36
8.4.2.6 OnGetMin()	36
8.4.2.7 OnInsert()	36
8.4.2.8 OnInsertManyItems()	36
8.4.2.9 OnKeyDownBackground()	37
8.4.2.10 OnKeyDownTextbox()	37
8.4.2.11 OnOpenStatsWindow()	37
8.4.2.12 SwitchAddItemsCount()	37
8.5 BinTreeVisualization.MainWindow Class Reference	38
8.5.1 Detailed Description	38
8.6 BinTreeVisualization.Algorithms.Node< T > Class Template Reference	38
8.6.1 Detailed Description	41
8.6.2 Constructor & Destructor Documentation	42
8.6.2.1 Node()	42
8.6.3 Member Function Documentation	42
8.6.3.1 ActivateBlue()	42
8.6.3.2 AdoptChild()	42
8.6.3.3 Blink()	43
8.6.3.4 CalcHeight()	43
8.6.3.5 CreateArrowTo()	43
8.6.3.6 CreateRoot< T >()	43
8.6.3.7 CreateUIElem()	44
8.6.3.8 DeactivateBlue()	44
8.6.3.9 GetDepth()	44
8.6.3.10 GetHeight()	45
8.6.3.11 GetLocOf()	45
8.6.3.12 GetNodeBalance()	45
8.6.3.13 GetRelativeNode()	45
8.6.3.14 IsLeaf()	46
8.6.3.15 MoveByLoc()	46
8.6.3.16 MoveChildrenByLoc()	46
8.6.3.17 MoveToLoc()	47
8.6.3.18 MoveToLocWithAnim()	47
8.6.3.19 MoveTreeByLoc()	47
8.6.3.20 MoveTreeToLoc()	47
8.6.3.21 operator<()	48
8.6.3.22 operator<=()	48
8.6.3.23 operator>()	48

8.6.3.24 operator>=()	49
8.6.3.25 OrphanChildren()	49
8.6.3.26 SpawnChild()	49
8.6.3.27 SwapValues()	50
8.6.3.28 Traverse()	50
8.6.3.29 TraverseAncestors()	50
8.6.4 Property Documentation	50
8.6.4.1 this[Side side]	50
8.7 BinTreeVisualization.UI.NodeArrow Class Reference	51
8.7.1 Detailed Description	53
8.7.2 Member Function Documentation	53
8.7.2.1 GetCanvas()	53
8.7.2.2 GetDistanceTo()	53
8.7.2.3 GetGoodSideAngle()	54
8.7.2.4 GetLeftArrowSocket()	54
8.7.2.5 GetLocOf()	54
8.7.2.6 GetLowerArrowSocket()	55
8.7.2.7 GetNodeBorder()	55
8.7.2.8 GetRightArrowSocket()	55
8.7.2.9 GetRotToTarget()	56
8.7.2.10 GetUpperArrowSocket()	56
8.7.2.11 MoveSourceToLoc()	56
8.7.2.12 MoveTargetToLoc()	56
8.7.2.13 RotateToLoc()	57
8.7.2.14 RotateToTarget()	57
8.7.3 Member Data Documentation	57
8.7.3.1 SourceProp	57
8.7.3.2 TargetProp	58
8.8 BinTreeVisualization.UI.NodeControl Class Reference	58
8.8.1 Detailed Description	59
8.8.2 Member Function Documentation	59
8.8.2.1 MoveToLoc()	59
8.8.2.2 Node_OnClick()	59
8.9 BinTreeVisualization.UI.PointExtensions Class Reference	61
8.9.1 Member Function Documentation	61
8.9.1.1 Normal()	61
8.9.1.2 Size()	61
8.10 BinTreeVisualization.UI.ProgressLabel Class Reference	62
8.10.1 Detailed Description	62
8.11 BinTreeVisualization.Stats.StatsWindow Class Reference	62
8.11.1 Detailed Description	63
8.11.2 Member Function Documentation	63

8.11.2.1 OnClosed()	63
8.12 BinTreeVisualization.Algorithms.TextActionColorsHelper Class Reference	63
8.12.1 Detailed Description	64
8.12.2 Member Function Documentation	64
8.12.2.1 GetColor()	64
8.13 BinTreeVisualization.Stats.TreeStats Class Reference	64
8.13.1 Detailed Description	65
8.13.2 Member Function Documentation	65
8.13.2.1 AddStats()	65
8.13.2.2 FillPlot()	66
8.13.2.3 GetList()	66
8.13.2.4 GetListWithAverages()	66
8.13.2.5 Refresh()	67
Index	69

Chapter 1

General assignment

Make a visualization of basic operations on BST and AVL trees using any programming language and UI framework. Operations to support:

- Search
- Insert
- Delete
- Find min
- Find max

Should ship with documentation that focuses on the properties of binary trees and used data structured, with documentation of the UI implementation being a secondary target.

1.1 Roadmap

Project requirements:

- [X] Initial UI setup
- [X] First block spawned
- [X] Setup text output
- [X] Setup animations
- [X] Setup arrows
- [X] Fill in algorithms
- [X] Make auto layout

Additional subobjectives:

- [X] Add more UI controls
- [X] Add statistics on operation count
- [X] Scale tree to fit automatically
- [] Add Red Black Trees?
- [] Save / load trees?
- [] Add tests

1.2 AVL tree in general

A nice writeup on AVL trees: <https://simpledevcode.wordpress.com/2014/09/16/avl-tree-in-c/>

It is mostly a BST tree with added rotations, so a single base class with a `bool bDoRotations` should suffice.

1.3 WPF specifics

- Arrows do not exist per-se as a WPF object. Can make a container with several lines to mimic them:
<https://stackoverflow.com/a/68552890>
- For centering nodes:
 - Grid supports putting its items in location relative to center. Canvas do not respect that by default.
 - Can work around that by putting Canvas in a Grid - its reflectiveness seems to be inherited.
- Animating locations of custom arrows require a `DependencyProp`, which then can be fed into `Point↔Animation`:
 - <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/properties/dependency-property>
 - <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/graphics-multimedia/how-to>
- Auto-scaling canvas should be possible with a `ViewBox`, but it clears out entire view on use?
 - Canvas do support transforms, so can just apply `ScaleTransformation` dynamically

1.4 Layouting nodes

- Layouting nodes of a binary tree in such a way so that neither space is wasted nor the nodes overlay seem to be harder than expected. Might shelve it for now.
- Found suggestion: Reingold-Tilford's algorithm - but it is for normal trees and not binary ones
- A working algorithm from 1981: <https://reingold.co/tidier-drawings.pdf> (Tidier Drawing of Trees by Edward M. Reingold and John S. Tilford (IEEE Transactions on Software Engineering, Volume 7, Number 2, March 1981))
- This paper also references an easier Wetherell and Shannon algorithm, which has some flaws (as highlighted in the paper), but it also easier to implement
 - Its cons seem to be insignificant enough for this application
 - Also highlighted here: <https://willrosenbaum.com/assets/pdf/2023s-cosc-225/tidy-drawing.pdf>

1.5 Performance

- Performance in instant mode is being capped by creating and running animations in the background
 - Especially by arrows whose locations are being computed on each movement.
- Optimizing for instant mode would require disabling the animations, but it is probably not worth it for the purposes of this application.

Chapter 2

Project overview

2.1 Base BST

The binary search tree is build out of nodes - starting by root, each of the subsequent children can have two children of their own.

The base property of the binary tree which allows for $O(\log n)$ search is the way the children nodes are placed - nodes smaller than their parent are placed on the left, and nodes bigger than their parent - on the right. Thanks to this, the area of tree which needs to be searched is effectively halved with each recursive iterations (assuming the tree is balanced well enough).

2.1.1 Insertion

- Proceed like with normal search, then if a spot is found where the search would continue but the spot is empty - insert a new node into it.

2.1.2 Search

- Follow the tree with the principle of going left if the searched value is smaller than the current node, and right if the searched value is bigger.
- Check if the current is the searched node if neither of those paths can be taken.

2.1.3 Deletion

- Find an item to delete.
- Next, try to find its successor - the next element which is minimally smaller than the current one (that can be obtained by going one to right and then max to left, thanks to binary tree properties).
- Once a successor is found, swap the victim with the sucesor, and then remove the victim whose been moved out of the original location.
- If the victim is not placed in a leaf position (i.e. having no children), repeat finding a successor and swapping until a leaf node is reached which can then be removed safely.

2.2 AVL

AVL tree is a type of a binary search tree. It operates on the same base principle regarding to the location of nodes, but with an additional mechanic - rotations.

Whenever a subtree becomes unbalanced - i.e. one of its sides is at least two nodes taller than the other one - a node with the middle value is picked out of the long branch, and then the other nodes from this branch are balanced around it.

Thanks to this, a normally successive chain (eg. 1 -> 2 -> 3) becomes a neatly balanced, smaller tree (with node {2} pointing to {1} to the left and to {3} to the right).

Operations on AVL trees start exactly the same way as on base BSTs, but then after a node is either inserted or removed, all parents of the affected node are visited to make sure the tree is kept in balance, and rotations are performed if they are not.

2.3 App structure

The application is written utilizing C# 13 preview language, .NET 9.0 framework, as well as WPF framework for the UI.

The tree is layouted using a modified Wetherell and Shannon algorithm.

It uses Material Design (<http://materialdesigninxaml.net/>) NuGet package for the interface theme, as well as ScottPlot library (<https://scottplot.net/>) for rendering the collected performance data into graphs.

The tree's logical data structures are partially separated from the WPF elements and use a composite model.

- `BinTree<T>` and `Node<T>` are the main data logical structures that hold the info about the binary tree and can perform operations on it.
- `TreeLayout` is a helper class that can auto-layout the tree.
- `BinTreeMan` is a WPF page which holds the main user interface and delegates operations requested by user to the binary tree.
- `BinTreeControl` is a WPF UserControl that holds Canvas where the nodes are physically placed into
- `NodeControl` is a WPF UserControl that represents a visual node - each `NodeControl` is directly managed by a `Node<T>` it is associated with, and physically placed upon `BinTreeControl`'s Canvas
- `NodeArrow` is a WPF UserControl that represents an arrow that can point from one point into another. Each `Node<T>` manages an arrow pointing towards it.
- `ProgressLabel` is a WPF Control that is used for displaying descriptions.
- `TreeStats` and `StatsWindow.xaml` are classes used for gathering and displaying the statistics respectively.

Documentation PDF is generated with Doxygen.

Detailed descriptions of each of the classes, methods, and properties used in the program are contained in the following chapters of the documentation PDF.

Chapter 3

Binary Tree Visualization

A modern C# / WPF application that lets you visualize basic operations on BST and AVL trees.

3.1 Operations supported

Supports:

- Insertion
- Deletion
- Search

All steps performed during an algorithm's execution are described in the log, and all the nodes associated with the particular sub-operation are highlighted - which enables clear and easy following of an algorithm.

Additionally, the amount of performed comparisons and traversals is recorded and can be viewed in form of a graph (utilizes ScottPlott library):

This allows one to verify the algorithms' time complexity with real-world data.

3.2 Quality of Lives

- Use "Insert 20 items" button to insert twenty random items into the tree instantly. Count can be decreased / increased with F2 and F3 keys respectively.
- Press `Escape` to cancel the current animation and proceed to the final result instantly
- Press `Enter` to repeat the last operation
- Click a node to quickly select it
- Graph's data updates instantly as you perform new operations
- The tree is auto-layed out using a modified Wetherell and Shannon algorithm for visually pleasant and efficient use of space

3.3 Build

Uses .NET 9.0 and C# 13 preview with WPF UI. To build with Visual Studio 2022.

Utilizes `Material Design` NuGet package for the interface theme, as well as `ScottPlot library` for rendering the collected performance data into graphs.

The interface is connected to the logical binary tree with a composite model, where `BinTree<T>` and `Node<T>` manage their respective `BinTreeControl` and `BinTreeNode` WPF controls, respectively.

Chapter 4

Namespace Index

4.1 Package List

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

BinTreeVisualization	13
BinTreeVisualization.Algorithms	13
BinTreeVisualization.Stats	14
BinTreeVisualization.UI	15

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

Application	
BinTreeVisualization.App	17
INotifyPropertyChanged	
BinTreeVisualization.Algorithms.BinTree< T >	17
BinTreeVisualization.UI.BinTreeMan	33
BinTreeVisualization.UI.NodeControl	58
Label	
BinTreeVisualization.UI.ProgressLabel	62
BinTreeVisualization.Algorithms.Node< T >	38
Page	
BinTreeVisualization.UI.BinTreeMan	33
BinTreeVisualization.UI.PointExtensions	61
BinTreeVisualization.Algorithms.TextActionColorsHelper	63
BinTreeVisualization.Stats.TreeStats	64
UserControl	
BinTreeVisualization.UI.BinTreeControl	29
BinTreeVisualization.UI.NodeArrow	51
BinTreeVisualization.UI.NodeControl	58
Window	
BinTreeVisualization.MainWindow	38
BinTreeVisualization.Stats.StatsWindow	62

Chapter 6

Class Index

6.1 Class List

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

BinTreeVisualization.App	
Interaction logic for App.xaml	17
Can function as a basic BST and an AVL tree	17
BinTreeVisualization.UI.BinTreeControl	
WPF UserControl that holds all binary tree's nodes	29
BinTreeVisualization.UI.BinTreeMan	
Page which holds both a BinTreeControl and other controls that allow the user to interact with it	33
BinTreeVisualization.MainWindow	
Main window which holds the BinTreeMan page	38
BinTreeVisualization.Algorithms.Node< T >	
A binary tree's node	38
BinTreeVisualization.UI.NodeArrow	
Represents an arrow that can point between nodes	51
BinTreeVisualization.UI.NodeControl	
UI element that represents a node in a binary tree	58
BinTreeVisualization.UI.PointExtensions	61
BinTreeVisualization.UI.ProgressLabel	
A label that can represent progress which supports spawn/despawn animations and colors . .	62
BinTreeVisualization.Stats.StatsWindow	
Interaction logic for StatsWindow.xaml	62
BinTreeVisualization.Algorithms.TextActionColorsHelper	
Helper for colors of a TextAction	63
BinTreeVisualization.Stats.TreeStats	
Holds statistics of operations performed on a binary tree	64

Chapter 7

Namespace Documentation

7.1 BinTreeVisualization Namespace Reference

Classes

- class [App](#)
Interaction logic for App.xaml.
- class [MainWindow](#)
Main window which holds the BinTreeMan page.

7.2 BinTreeVisualization.Algorithms Namespace Reference

Classes

- class [BinTree](#)
*A binary search tree, associated with a WPF backing control that nodes are being represented on.
Can function as a basic BST and an AVL tree.*
- class [Node](#)
A binary tree's node.
- class **SideExtensions**
Extensions for Side
- class [TextActionColorsHelper](#)
Helper for colors of a TextAction
- class **TreeLayout**
Helper class that layouts a binary tree so that no nodes overlap.

Enumerations

- enum [TextAction](#) { **Base** , **Blink** , **Violet** , **Red** }
Colors supported by the UI.
- enum [Side](#) { **Left** , **Right** }
Represents a side of a binary tree node.

Functions

- record struct [BinTreeRow< T >](#) (int Tier, List< [Node< T >](#) > Nodes)
A row in the binary tree, containing the tier and the nodes on that tier.

7.2.1 Function Documentation

7.2.1.1 [BinTreeRow< T >\(\)](#)

```
record struct BinTreeVisualization.Algorithms.BinTreeRow< T > (
    int Tier,
    List< Node< T > > Nodes )
```

A row in the binary tree, containing the tier and the nodes on that tier.

Template Parameters

<i>T</i>	
----------	--

Parameters

<i>Tier</i>	Tier
<i>Nodes</i>	All nodes in that tier

Type Constraints

T : *Comparable*<*T*>

7.3 BinTreeVisualization.Stats Namespace Reference

Classes

- class [StatsWindow](#)
Interaction logic for StatsWindow.xaml.
- class [TreeStats](#)
Holds statistics of operations performed on a binary tree.

Enumerations

- enum [OperationType](#) { **Insert** , **Delete** , **Search** , **Discard** }
Represents the type of operation performed on a binary tree.

Functions

- record struct [OperationStats](#) (double Comparisons, double Traversals, int TreeSize)
Represents the statistics of a single operation.

7.3.1 Function Documentation

7.3.1.1 OperationStats()

```
record struct BinTreeVisualization.Stats.OperationStats (
    double Comparisons,
    double Traversals,
    int TreeSize )
```

Represents the statistics of a single operation.

Parameters

<i>Comparisons</i>	The amount of comparisons between nodes' values
<i>Traversals</i>	The amount of travels from one node to another
<i>TreeSize</i>	The count of items in the tree at the beginning of the operation

7.4 BinTreeVisualization.UI Namespace Reference

Classes

- class [BinTreeControl](#)
WPF UserControl that holds all binary tree's nodes.
- class [BinTreeMan](#)
Page which holds both a BinTreeControl and other controls that allow the user to interact with it.
- class [NodeArrow](#)
Represents an arrow that can point between nodes.
- class [NodeControl](#)
UI element that represents a node in a binary tree.
- class [PointExtensions](#)
- class [ProgressLabel](#)
A label that can represent progress which supports spawn/despawn animations and colors.
- class **TitleBarHelper**
Helps to setup the title bar color of a window.

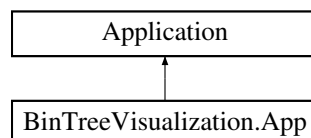
Chapter 8

Class Documentation

8.1 BinTreeVisualization.App Class Reference

Interaction logic for App.xaml.

Inheritance diagram for BinTreeVisualization.App:



8.1.1 Detailed Description

Interaction logic for App.xaml.

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

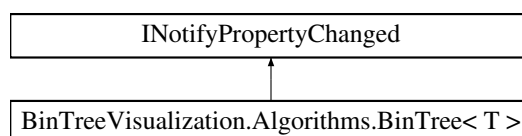
- S:/Uni/Algosity/BinTreeVisualization/App.xaml.cs

8.2 BinTreeVisualization.Algorithms.BinTree< T > Class Template Reference

A binary search tree, associated with a WPF backing control that nodes are being represented on.

Can function as a basic BST and an AVL tree.

Inheritance diagram for BinTreeVisualization.Algorithms.BinTree< T >:



Public Member Functions

- Canvas [GetCanvas](#) ()
Ref to the backing control's canvas.
- int [GetHeight](#) ()
Get the tree's height.
- async Task< bool > [VerifyTreesNotEmpty](#) ()
Verify whether the tree is not empty.
- async Task< [Node](#)< T >?> [Find](#) (T value)
Find value in the tree.
- async Task< [Node](#)< T >?> [Find](#) (T value, [Node](#)< T > currNode)
Recursively search the tree to find the specified value . Search starts from currNode .
- async Task< T > [GetMin](#) ()
Get minimum value in the tree.
- async Task< T > [GetMax](#) ()
Get maximum value in the tree.
- async Task< [Node](#)< T > > [GetMin](#) ([Node](#)< T > treeBase)
Get minimum in the subtree starting by the specified node.
- async Task< [Node](#)< T > > [GetMax](#) ([Node](#)< T > treeBase)
Get max in the subtree starting by the specified node.
- async void [Delete](#) (T value)
Delete node containing the specified value from the tree.
- void [VerifyTreeLayout](#) (bool fromMiddleOfOperation=false)
Verify that the tree's layout is correct: i.e. no nodes overlap and the scale is appropriate for all nodes to fit in the window.
- async Task [SwapValues](#) ([Node](#)< T > one, [Node](#)< T > other)
Swap nodes' values with animation.
- async Task [Delete](#) ([Node](#)< T > victim)
*Delete specified node from the tree.
You can find the node first with Find(T)*
- void **BreakInto** ()
Triggers debug break inside the tree.
- async Task [Insert](#) (T value)
Insert value into the tree.
- delegate void **OnInstantModeFinishedFunc** ()
Signature for OnInstantModeFinished
- async Task **FinishCurrOperation** ()
Instantly finish current operation, if any is active.
- async Task< [Node](#)< T > > [Insert](#) (T value, [Node](#)< T > currNode)
Internal insertion. Try to insert the value into the subtree starting with currNode .
- IEnumerable< [Node](#)< T > > **Traverse** ()
- List< BinTreeRow< T > > [GetRows](#) ()
Get a list of all rows in the tree with their depths.
- List< [Node](#)< T > > [GetRow](#) (int Depth)
Get a row of nodes at the specified depth.
- void **AssertTreesValid** ()
Assert that the tree is valid.
- void **AssertNoOrphans** ()
Assert that there are no orphaned nodes.
- void **AssertIsBinaryTree** ()
Assert that the core property of the binary tree (lesser children on the left, bigger children on the right) is satisfied for every node in the tree.

- void **AssertCachedHeights** ()
Assert that call cached heights are correct with the real heights.
- void **AssertIsAVLTree** ()
Assert that it is a correct AVL tree, i.e. at no point is branch balance more than 1.

Protected Member Functions

- virtual void **OnPropertyChanged** ([CallerMemberName] string propertyName="")

Properties

- **Node**< T >? **Root** [get, set]
Tree's root node.
- required **BinTreeControl** **BackingControl** [get]
The UI control associated with the tree which holds all the visual nodes, arrows, and progress text.
- int **Height** [get]
Tree's height.
- **TreeStats** **Stats** = new() [get, private set]
Statistics on how many operations did each operation on the tree require.
- bool **bSkipAnimations** = false [get, private set]
Whether or not to skip animations and perform operations instantly.
- SemaphoreSlim **OperationSem** = new(1, 1) [get]
Semaphore to ensure only one operation is being performed at a time.
- int **CurrWaiting** = 0 [get, set]
Number of tasks waiting for the operation to finish.
- int **CurrInside** = 0 [get, set]
Number of tasks currently performing an operation. Should never be bigger than one!
- bool **PerformRotations** = true [get, set]
Whether or not to perform AVL tree rotations.

Events

- **OnInstantModeFinishedFunc** **OnInstantModeFinished**
Event that is invoked when bSkipAnimations is switched back to false.
- PropertyChangedEventHandler **PropertyChanged**

Private Member Functions

- void **FinishOperationStats** (OperationType type)
*Coalesce gathered statistics into an **OperationStats** with the specified type and append to the tree's Stats.
Resets current operation's stats.*
- void **CreateRoot** (T value)
Create tree's root with the specified value .
- async Task< **Node**< T >?> **Find** (T value, bool bAsSupportingOp)
Find value in the tree.
- async Task **Blink** (**Node**< T > node)
Blink a node for 500ms and return once the blink starts to fade.
- async Task **SwapNodes** (**Node**< T > one, **Node**< T > other)
Swap two nodes in the tree and relink their parents in a way so that the binary tree properties are maintained.

- async Task [Delay](#) (int ms)
Delays the execution of the current task if in not-instant context. Otherwise, returns immediately.
- async Task [OperationGuard](#) ()
Ensures only one operation is being performed at a time. If one is already being performed when this func is called, try to skip all of its delays and wait for it to finish.
- void **FinishOperation** ()
Signal that an operation is finished and another one can be started.
- async Task [BalanceTreeIfNeeded](#) (Node< T > currNode)
Check if the subtree starting with the passed node is not skewed to a side, and perform a rotation if it is.
- async Task [AnimAdoption](#) (Node< T > parent, Node< T > child)
Perform adoption of a node by a parent with a 3-second long animation.
- async Task< Node< T > > [Rotate](#) (Node< T > currNode, bool bLeftRotation)
Side-heavy subtree. Rotate to the specified direction. The subtree should be rotated left if it's right-heavy (bLeftRotation = true), and right in the opposite case.
- async Task< Node< T > > [RotateRight](#) (Node< T > currNode)
Left-heavy subtree. Rotate middle nodes to the right.
- async Task< Node< T > > [RotateLeft](#) (Node< T > currNode)
Right-heavy subtree. Rotate middle nodes to the right.
- void **SetText** (string text, [TextAction](#) act=TextAction.Base)
- async Task **ResetText** ()
- void **LayoutTree** ()
Perform auto layout on the tree so that the nodes do not overlap with each other.

Private Attributes

- int **ComparisonsCount** = 0
The count of comparisons between nodes' values which were performed during the last operation.
- int **TraversalCount** = 0
The count of traversal between nodes (going up and down the tree) which were performed during the last operation.
- bool **bLayoutNSquare** = false
Whether to layout the tree in a square manner.

8.2.1 Detailed Description

A binary search tree, associated with a WPF backing control that nodes are being represented on.

Can function as a basic BST and an AVL tree.

Template Parameters

<i>T</i>	The type of data this tree's nodes hold
----------	---

Type Constraints

***T* : *Comparable*< *T*>**

8.2.2 Member Function Documentation

8.2.2.1 AnimAdoption()

```
async Task BinTreeVisualization.Algorithms.BinTree< T >.AnimAdoption (
    Node< T > parent,
    Node< T > child ) [private]
```

Perform adoption of a node by a parent with a 3-second long animation.

Parameters

<i>parent</i>	The new parent
<i>child</i>	The child to be adopted

Returns

8.2.2.2 BalanceTreeIfNeeded()

```
async Task BinTreeVisualization.Algorithms.BinTree< T >.BalanceTreeIfNeeded (
    Node< T > currNode ) [private]
```

Check if the subtree starting with the passed node is not skewed to a side, and perform a rotation if it is.

Parameters

<i>currNode</i>	Subtree which starts by this node
-----------------	-----------------------------------

Returns

8.2.2.3 Blink()

```
async Task BinTreeVisualization.Algorithms.BinTree< T >.Blink (
    Node< T > node ) [private]
```

Blink a node for 500ms and return once the blink starts to fade.

Parameters

<i>node</i>	Node to blink
-------------	---------------

Returns

8.2.2.4 CreateRoot()

```
void BinTreeVisualization.Algorithms.BinTree< T >.CreateRoot (
    T value ) [private]
```

Create tree's root with the specified *value* .

Parameters

<i>value</i>	Value to create the root with
--------------	-------------------------------

8.2.2.5 Delay()

```
async Task BinTreeVisualization.Algorithms.BinTree< T >.Delay (
    int ms ) [private]
```

Delays the execution of the current task if in not-instant context.

Otherwise, returns immediately.

Parameters

<i>ms</i>	Delay in milliseconds
-----------	-----------------------

Returns

8.2.2.6 Delete() [1/2]

```
async Task BinTreeVisualization.Algorithms.BinTree< T >.Delete (
    Node< T > victim )
```

Delete specified node from the tree.

You can find the node first with Find(T)

Parameters

<i>victim</i>	Node to delete
---------------	----------------

Returns**8.2.2.7 Delete()** [2/2]

```
async void BinTreeVisualization.Algorithms.BinTree< T >.Delete (
    T value )
```

Delete node containing the specified value from the tree.

Parameters

<i>value</i>	The value to remove
--------------	---------------------

8.2.2.8 Find() [1/3]

```
async Task< Node< T >? > BinTreeVisualization.Algorithms.BinTree< T >.Find (
    T value )
```

Find *value* in the tree.

Parameters

<i>value</i>	The value to find
--------------	-------------------

Returns

The found node. null if node with the requested value was not found.

8.2.2.9 Find() [2/3]

```
async Task< Node< T >? > BinTreeVisualization.Algorithms.BinTree< T >.Find (
    T value,
    bool bAsSupportingOp ) [private]
```

Find *value* in the tree.

Parameters

<i>value</i>	The value to find
<i>bAsSupportingOp</i>	true if it's executed as part of another operation and thus should not wait for exclusivity

Returns

The found node. null if node with the requested value was not found.

8.2.2.10 Find() [3/3]

```
async Task< Node< T >?> BinTreeVisualization.Algorithms.BinTree< T >.Find (
    T value,
    Node< T > currNode )
```

Recursively search the tree to find the specified *value* . Search starts from *currNode* .

Parameters

<i>value</i>	The value to find.
<i>currNode</i>	Subtree to search for the node in currently.

Returns

The found node. null if node with the requested value was not found.

8.2.2.11 FinishOperationStats()

```
void BinTreeVisualization.Algorithms.BinTree< T >.FinishOperationStats (
    OperationType type ) [private]
```

Coalesce gathered statistics into an [OperationStats](#) with the specified *type* and append to the tree's Stats.

Resets current operation's stats.

Parameters

<i>type</i>	The type of operation which was performed
-------------	---

8.2.2.12 GetCanvas()

```
Canvas BinTreeVisualization.Algorithms.BinTree< T >.GetCanvas ( )
```

Ref to the backing control's canvas.

Returns**8.2.2.13 GetHeight()**

```
int BinTreeVisualization.Algorithms.BinTree< T >.GetHeight ( )
```

Get the tree's height.

Returns

8.2.2.14 GetMax() [1/2]

```
async Task< T > BinTreeVisualization.Algorithms.BinTree< T >.GetMax ( )
```

Get maximum value in the tree.

Returns

The max value in the tree

8.2.2.15 GetMax() [2/2]

```
async Task< Node< T > > BinTreeVisualization.Algorithms.BinTree< T >.GetMax (
    Node< T > treeBase )
```

Get max in the subtree starting by the specified node.

Parameters

<i>treeBase</i>	Tree's base from which the search should start
-----------------	--

Returns

The node with the maximum value in the subtree

8.2.2.16 GetMin() [1/2]

```
async Task< T > BinTreeVisualization.Algorithms.BinTree< T >.GetMin ( )
```

Get minimum value in the tree.

Returns

The min value in the tree

8.2.2.17 GetMin() [2/2]

```
async Task< Node< T > > BinTreeVisualization.Algorithms.BinTree< T >.GetMin (
    Node< T > treeBase )
```

Get minimum in the subtree starting by the specified node.

Parameters

<i>treeBase</i>	Tree's base from which the search should start
-----------------	--

Returns

The node with the minimum value in the subtree

8.2.2.18 GetRow()

```
List< Node< T > > BinTreeVisualization.Algorithms.BinTree< T >.GetRow (
    int Depth )
```

Get a row of nodes at the specified depth.

Parameters

<i>Depth</i>	The depth
--------------	-----------

Returns

A list of all nodes at the specified depth.

8.2.2.19 GetRows()

```
List< BinTreeRow< T > > BinTreeVisualization.Algorithms.BinTree< T >.GetRows ( )
```

Get a list of all rows in the tree with their depths.

Returns

A list of all rows in the tree with their depths.

8.2.2.20 Insert() [1/2]

```
async Task BinTreeVisualization.Algorithms.BinTree< T >.Insert (
    T value )
```

Insert *value* into the tree.

Parameters

<i>value</i>	The value to insert.
--------------	----------------------

8.2.2.21 Insert() [2/2]

```
async Task< Node< T > > BinTreeVisualization.Algorithms.BinTree< T >.Insert (
    T value,
    Node< T > currNode )
```

Internal insertion. Try to insert the *value* into the subtree starting with *currNode* .

Parameters

<i>value</i>	The value to insert.
<i>currNode</i>	Subtree which starts by this node.

Returns

Passed-in subtree with inserted *value*

8.2.2.22 OperationGuard()

```
async Task BinTreeVisualization.Algorithms.BinTree< T >.OperationGuard ( ) [private]
```

Ensures only one operation is being performed at a time.

If one is already being performed when this func is called, try to skip all of its delays and wait for it to finish.

Returns

8.2.2.23 Rotate()

```
async Task< Node< T > > BinTreeVisualization.Algorithms.BinTree< T >.Rotate (
    Node< T > currNode,
    bool bLeftRotation ) [private]
```

Side-heavy subtree. Rotate to the specified direction.

The subtree should be rotated left if it's right-heavy (*bLeftRotation* = `true`), and right in the opposite case.

Parameters

<i>currNode</i>	Node which the unbalanced subtree starts with
<i>bLeftRotation</i>	<code>true</code> if the subtree is right-heavy and should be rotated left.

`false` if the subtree is left-heavy and should be rotated right

Returns

New top child who the subtree starts with

8.2.2.24 RotateLeft()

```
async Task< Node< T > > BinTreeVisualization.Algorithms.BinTree< T >.RotateLeft (
    Node< T > currNode ) [private]
```

Right-heavy subtree. Rotate middle nodes to the right.

Parameters

<i>currNode</i>	Node which the unbalanced subtree starts with
-----------------	---

Returns

New top child who the subtree starts with

8.2.2.25 RotateRight()

```
async Task< Node< T > > BinTreeVisualization.Algorithms.BinTree< T >.RotateRight (
    Node< T > currNode ) [private]
```

Left-heavy subtree. Rotate middle nodes to the right.

Parameters

<i>currNode</i>	Node which the unbalanced subtree starts with
-----------------	---

Returns

New top child who the subtree starts with

8.2.2.26 SwapNodes()

```
async Task BinTreeVisualization.Algorithms.BinTree< T >.SwapNodes (
    Node< T > one,
    Node< T > other ) [private]
```

Swap two nodes in the tree and relink their parents in a way so that the binary tree properties are maintained.

Parameters

<i>one</i>	One node
<i>other</i>	The other node

8.2.2.27 SwapValues()

```
async Task BinTreeVisualization.Algorithms.BinTree< T >.SwapValues (
    Node< T > one,
    Node< T > other )
```

Swap nodes' values with animation.

Parameters

<i>one</i>	One node
<i>other</i>	The other node

Returns

8.2.2.28 VerifyTreeIsNotEmpty()

```
async Task< bool > BinTreeVisualization.Algorithms.BinTree< T >.VerifyTreeIsNotEmpty ( )
```

Verify whether the tree is not empty.

Returns

`true` if the tree is not empty, `false` otherwise

8.2.2.29 VerifyTreeLayout()

```
void BinTreeVisualization.Algorithms.BinTree< T >.VerifyTreeLayout (
    bool fromMiddleOfOperation = false )
```

Verify that the tree's layout is correct: i.e. no nodes overlap and the scale is appropriate for all nodes to fit in the window.

Parameters

<i>fromMiddleOfOperation</i>	<code>true</code> it is called from middle of an operation, and <code>false</code> if it's called after operations are finished
------------------------------	---

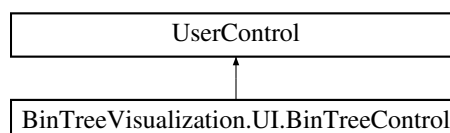
The documentation for this class was generated from the following file:

- S:/Uni/Algosity/BinTreeVisualization/Algorithms/BinTree.cs

8.3 BinTreeVisualization.UI.BinTreeControl Class Reference

WPF UserControl that holds all binary tree's nodes.

Inheritance diagram for BinTreeVisualization.UI.BinTreeControl:



Public Member Functions

- void [SetScaleAnim](#) (double newScale)
Set the canvas scale with an animation.
- void [VerifyScale](#)< T > ([BinTree](#)< T > tree)
Check whether a new scale is needed for all elements to fit.
- void [SetText](#) (string text, [TextAction](#) act=TextAction.Base)
Add a new text describing the current step. The newly spawned text is animated.
- async Task [ResetText](#) (bool bWait)
Reset all current progress texts.
- void [LayoutTreeNSquare](#)< T > ([BinTree](#)< T > tree)
Auto layout the tree, allocating double the space for every next row. Does not account for empty branches.

Properties

- double **currVirtualWidth** [get]
The amount of real space that the tree can accumulate at the current scale.

Private Member Functions

- double [GetNewScale](#)< T > ([BinTree](#)< T > tree)
Get the new scale for the tree to fit the window.
- double [GetNodesTotalWidth](#)< T > ([BinTree](#)< T > tree)
Get the maximum spread between the westmost and the eastmost nodes.

Private Attributes

- double **CanvasDesiredScale** = 1.0
Current canvas desired scale after animations are finished.
- double **RescaleMultiplier** = 1.3
Unused.
- double **MinScale** = 1
Minimum scale of the tree. Do not make it smaller than that.

8.3.1 Detailed Description

WPF UserControl that holds all binary tree's nodes.

8.3.2 Member Function Documentation

8.3.2.1 [GetNewScale](#)< T >()

```
double BinTreeVisualization.UI.BinTreeControl.GetNewScale< T > (
    BinTree< T > tree ) [private]
```

Get the new scale for the tree to fit the window.

Template Parameters

<i>T</i>	
----------	--

Parameters

<i>tree</i>	The tree
-------------	----------

Returns

A new scale for the canvas

Type Constraints

***T* : *Comparable*<*T*>**

8.3.2.2 GetNodesTotalWidth< T >()

```
double BinTreeVisualization.UI.BinTreeControl.GetNodesTotalWidth< T > (
    BinTree< T > tree ) [private]
```

Get the maximum spread between the westmost and the eastmost nodes.

Template Parameters

<i>T</i>	
----------	--

Parameters

<i>tree</i>	The tree
-------------	----------

Returns

Spread of the tree, rounded up so that the tree stays symmetrical.

Type Constraints

***T* : *Comparable*<*T*>**

8.3.2.3 LayoutTreeNSquare< T >()

```
void BinTreeVisualization.UI.BinTreeControl.LayoutTreeNSquare< T > (
    BinTree< T > tree )
```

Auto layout the tree, allocating double the space for every next row. Does not account for empty branches.

Template Parameters

<i>T</i>	
----------	--

Parameters

<i>tree</i>	The tree
-------------	----------

Type Constraints

***T* : *Comparable*<*T*>**

8.3.2.4 ResetText()

```
async Task BinTreeVisualization.UI.BinTreeControl.ResetText (
    bool bWait )
```

Reset all current progress texts.

If any texts are currently present, they will be despawned in a 200ms long animation.

If empty, returns immediately.

8.3.2.5 SetScaleAnim()

```
void BinTreeVisualization.UI.BinTreeControl.SetScaleAnim (
    double newScale )
```

Set the canvas scale with an animation.

Parameters

<i>newScale</i>	The new target scale to be set to after the animation finishes.
-----------------	---

8.3.2.6 SetText()

```
void BinTreeVisualization.UI.BinTreeControl.SetText (
    string text,
    TextAction act = TextAction::Base )
```

Add a new text describing the current step. The newly spawned text is animated.

Parameters

<i>text</i>	The text to add
<i>act</i>	Text's color

8.3.2.7 VerifyScale< T >()

```
void BinTreeVisualization.UI.BinTreeControl.VerifyScale< T > (
    BinTree< T > tree )
```

Check whether a new scale is needed for all elements to fit.

Template Parameters

<i>T</i>	
----------	--

Parameters

<i>tree</i>	The tree
-------------	----------

Type Constraints

***T* : *Comparable*< *T*>**

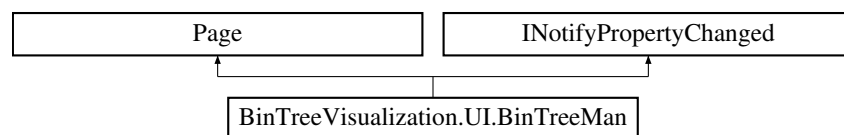
The documentation for this class was generated from the following file:

- S:/Uni/Algosy/BinTreeVisualization/UI/BinTreeControl.xaml.cs

8.4 BinTreeVisualization.UI.BinTreeMan Class Reference

Page which holds both a BinTreeControl and other controls that allow the user to interact with it.

Inheritance diagram for BinTreeVisualization.UI.BinTreeMan:



Protected Member Functions

- virtual void **OnPropertyChanged** ([CallerMemberName] string propertyName="")

Properties

- **BinTree**< double > **BinTree** [get, set]
The associated binary tree whose data is being displayed.
- string **OperationArgument** [get, set]
The argument of the operation to be performed.
- bool **PerformRotations** = true [get, set]
Backing bool for whether or not to perform AVL rotations.
- int **AddItemsCount** = 20 [get, set]
Backing field for the count of items to add.
- string **AddItemsCountText** [get]
UI text for the "Insert X items" button.

Events

- PropertyChangedEventHandler **PropertyChanged**

Private Member Functions

- async void **Insert** (double value)
- async void **OnInsert** (object sender, RoutedEventArgs e)
Click event - perform an insert operation.
- void **OnGetMin** (object sender, RoutedEventArgs e)
Click event - on get min.
- void **OnGetMax** (object sender, RoutedEventArgs e)
Click event - on get max.
- bool **GetDoubleArg** (out double arg)
Try to get the current InputTextBox's value as a double.
- void **OnFind** (object sender, RoutedEventArgs e)
Click event - on find.
- void **OnDelete** (object sender, RoutedEventArgs e)
Click event - on delete.
- void **SwitchAddItemsCount** (bool bToDown)
Switch the value of AddItemsCount to the next or previous value.
- async void **AddRandomItems** (int count)
Click event - on add random items.
- void **OnKeyDownTextbox** (object sender, KeyEventArgs e)
On key pressed inside the InputTextBox.
- void **OnKeyDownBackground** (object sender, KeyEventArgs e)
On key pressed inside the entire main window.
- void **OnInsertManyItems** (object sender, RoutedEventArgs e)
Click event - on insert many items.
- void **OnOpenStatsWindow** (object sender, RoutedEventArgs e)
Click event - on clear.
- void **Page_SizeChanged** (object sender, SizeChangedEventArgs e)

Private Attributes

- List< int > **AddItemsCounts** = [10, 20, 50, 100, 200]
List of possible values for AddItemsCount.
- Action< object, RoutedEventArgs > **lastOperation**
The last operation that was performed. It will be executed when pressing the Enter key.

8.4.1 Detailed Description

Page which holds both a BinTreeControl and other controls that allow the user to interact with it.

8.4.2 Member Function Documentation

8.4.2.1 AddRandomItems()

```
async void BinTreeVisualization.UI.BinTreeMan.AddRandomItems (
    int count ) [private]
```

Click event - on add random items.

Parameters

<i>count</i>	The amount of random items to add
--------------	-----------------------------------

8.4.2.2 GetDoubleArg()

```
bool BinTreeVisualization.UI.BinTreeMan.GetDoubleArg (
    out double arg ) [private]
```

Try to get the current InputTextBox's value as a double.

Parameters

<i>arg</i>	Out arg: the parsed value
------------	---------------------------

Returns

`true` if parse succeeded, `false` otherwise.

8.4.2.3 OnDelete()

```
void BinTreeVisualization.UI.BinTreeMan.OnDelete (
    object sender,
    RoutedEventArgs e ) [private]
```

Click event - on delete.

Parameters

<i>sender</i>	
<i>e</i>	

8.4.2.4 OnFind()

```
void BinTreeVisualization.UI.BinTreeMan.OnFind (
    object sender,
    RoutedEventArgs e ) [private]
```

Click event - on find.

Parameters

<i>sender</i>	
<i>e</i>	

8.4.2.5 OnGetMax()

```
void BinTreeVisualization.UI.BinTreeMan.OnGetMax (
    object sender,
    RoutedEventArgs e ) [private]
```

Click event - on get max.

Parameters

<i>sender</i>	
<i>e</i>	

8.4.2.6 OnGetMin()

```
void BinTreeVisualization.UI.BinTreeMan.OnGetMin (
    object sender,
    RoutedEventArgs e ) [private]
```

Click event - on get min.

Parameters

<i>sender</i>	
<i>e</i>	

8.4.2.7 OnInsert()

```
async void BinTreeVisualization.UI.BinTreeMan.OnInsert (
    object sender,
    RoutedEventArgs e ) [private]
```

Click event - perform an insert operation.

Parameters

<i>sender</i>	
<i>e</i>	

8.4.2.8 OnInsertManyItems()

```
void BinTreeVisualization.UI.BinTreeMan.OnInsertManyItems (
    object sender,
    RoutedEventArgs e ) [private]
```

Click event - on insert many items.

Parameters

<i>sender</i>	
<i>e</i>	

8.4.2.9 OnKeyDownBackground()

```
void BinTreeVisualization.UI.BinTreeMan.OnKeyDownBackground (
    object sender,
    KeyEventArgs e ) [private]
```

On key pressed inside the entire main window.

Parameters

<i>sender</i>	
<i>e</i>	

8.4.2.10 OnKeyDownTextbox()

```
void BinTreeVisualization.UI.BinTreeMan.OnKeyDownTextbox (
    object sender,
    KeyEventArgs e ) [private]
```

On key pressed inside the InputTextBox.

Parameters

<i>sender</i>	
<i>e</i>	

8.4.2.11 OnOpenStatsWindow()

```
void BinTreeVisualization.UI.BinTreeMan.OnOpenStatsWindow (
    object sender,
    RoutedEventArgs e ) [private]
```

Click event - on clear.

Parameters

<i>sender</i>	
<i>e</i>	

8.4.2.12 SwitchAddItemsCount()

```
void BinTreeVisualization.UI.BinTreeMan.SwitchAddItemsCount (
```

```
bool bToDown ) [private]
```

Switch the value of AddItemsCount to the next or previous value.

Parameters

<i>bToDown</i>	Whether to switch the value down
----------------	----------------------------------

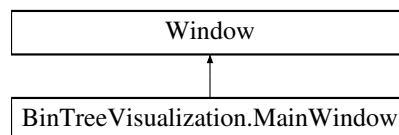
The documentation for this class was generated from the following file:

- S:/Uni/Algosity/BinTreeVisualization/UI/BinTreeMan.xaml.cs

8.5 BinTreeVisualization.MainWindow Class Reference

Main window which holds the BinTreeMan page.

Inheritance diagram for BinTreeVisualization.MainWindow:



8.5.1 Detailed Description

Main window which holds the BinTreeMan page.

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

- S:/Uni/Algosity/BinTreeVisualization/UI/MainWindow.xaml.cs

8.6 BinTreeVisualization.Algorithms.Node< T > Class Template Reference

A binary tree's node.

Public Member Functions

- void **RefreshHeight** ()
Refresh node height. To be used after adoptions.
- [Node< T > SpawnChild](#) (T value, bool bLeft)
Spawn a new node as child of this node. This is the only way to spawn new nodes.
- bool [IsLeaf](#) ()
Whether the node is a leaf (i.e. has no children)
- [Node< T > AdoptChild](#) ([Node< T > node](#))
*Adopt a child node. The child along its entire subtree will be moved to the correct side of this node.
If selves children slots are full, the passed-in node will itself adopt one of the current children in order to create a chain.*
- void [OrphanChildren](#) (bool OrphanLeftChild=true, bool OrphanRightChild=true)
Orphan the children of this node.
- void [SwapValues](#) ([Node< T > other](#))
Swap value of this node with value from another one. Does NOT ensure binary tree properties are maintained!
- void **DetachFromParent** ()
Detach this node from its parent.
- int [GetDepth](#) ()
Get depth, i.e. how many levels deep is this node in the tree.
- int [GetHeight](#) ()
Get height of the node, i.e. how many levels deep is the deepest child of this node.
- int [CalcHeight](#) ()
Calculate node's height recursively.
- int [GetNodeBalance](#) ()
Get the balance of the node - how much taller is its left children trace than the right one.
- [Node< T > GetRelativeNode](#) (int toRight)
*Get the node at a relative position to the right from the current node, as if all nodes on this level were taken.
Returns null if the spot at this location is empty.*
- [IEnumerable< Node< T > > Traverse](#) ()
Traverse the tree in Level Order Traversal.
- [IEnumerable< Node< T > > TraverseAncestors](#) ()
Traverse ancestors of this node in the order parent -> grandparent -> ... -> root.
- override string **ToString** ()
- void **RefreshSelfArrow** ()
Refresh the arrow pointing to this node to account for changes in parents.
- void **Activate** ()
- void **Deactivate** ()
- void **HighlightBlue** ()
- void [ActivateBlue](#) (bool bRecursive=false)
Change node's fill color to blue with an animation.
- void [DeactivateBlue](#) (bool bRecursive=false)
Reset node's fill color.
- void [Blink](#) (bool bRecursive=false)
Blink the node's UI control in blue.
- void [MoveTreeToLoc](#) (Point loc, bool bDelayAnimation=false)
Move the associated node control to a location and make its all its children follow it.
- void [MoveTreeByLoc](#) (Point deltaLoc, bool bDelayAnimation=false)
Move the associated node control by a delta location and make its all its children follow it.
- void [MoveChildrenByLoc](#) (Point deltaLoc, bool bDelayAnimation=false)
Move all the children of this node (but NOT this node itself) by a delta location.
- void [MoveByLoc](#) (Point loc, bool bDelayAnimation=false)

- *Move the associated node control by a specified amount.*
- void **MoveToLoc** (Point loc, bool bDelayAnimation=false)
Move the associated node control to specified location over 0.5 seconds.
- void **MoveToLocWithAnim** (Point loc, bool bDelayAnimation=false)
Move the associated node control to specified location over 0.5 seconds. Always play the animation - regardless of instant mode or others.
- void **PlayDelayedAnimation** ()
Play cached animation now. Reset the cache state.

Static Public Member Functions

- static **Node**< T > **CreateRoot**< T > (T value, **BinTree**< T > tree)
Create a root node to start a new tree.
- static bool **operator**< (**Node**< T > self, **Node**< T > other)
Compare nodes value-wise.
- static bool **operator**> (**Node**< T > self, **Node**< T > other)
Compare nodes value-wise.
- static bool **operator**< (T self, **Node**< T > other)
- static bool **operator**> (T self, **Node**< T > other)
- static bool **operator**<= (**Node**< T > self, **Node**< T > other)
Compare nodes value-wise.
- static bool **operator**>= (**Node**< T > self, **Node**< T > other)
Compare nodes value-wise.
- static bool **operator**<= (T self, **Node**< T > other)
- static bool **operator**>= (T self, **Node**< T > other)

Static Public Attributes

- const int **ToBottomOffset** = 70
Offset by which nodes should be separated vertically.
- const int **ToSideOffset** = 100
Offset by which children nodes should be put to side by from their parent nodes horizontally..
- const double **NodeWidth** = 120
Node width.
- const double **NodeHeight** = 70
Node height.

Properties

- **BinTree**< T > **Tree** [get]
Node's tree.
- **Node**< T >? **Parent** [get, set]
The node's parent.
- T **Value** = null [get, private set]
The node's value.
- **Node**< T >? **Left** = null [get, private set]
The left child. Use AdoptChild(Node<T>) or OrphanChildren(bool, bool) to change this reference.
- **Node**< T >? **Right** = null [get, private set]
The right child. Use AdoptChild(Node<T>) or OrphanChildren(bool, bool) to change this reference.

- int **Height** = 1 [get, private set]
Cached node height. Can be refreshed with RefreshHeight
- [Node< T >? this\[Side side\]](#) [get, set]
Get or set a node by its side.
- required [NodeControl](#) **BackingControl** [get]
Backing element placed in the UI.
- [NodeArrow? SelfArrow](#) = null [get, private set]
UI arrow pointing to this node.
- double **CurrWidth** [get]
Current width of the UI control.
- double **CurrHeight** [get]
Current height of the UI control.
- Point **DesiredLoc** [get, private set]
*Desired location of the node. Might be either the current location, or the location the new will finish at once its animation end.
Use MoveToLoc(Point) to move the node to a new location.*
- Point **CurrLoc** [get]
*Current location of the node in the UI. Might be volatile because it is affected by animations.
Use DesiredLoc for location-related operations.*
- bool **HasCachedDesiredLoc** = false [get, set]
Whether the node has cached loc that it should move to with PlayDelayedAnimation once available.

Private Member Functions

- [Node](#) (T value, [BinTree< T > tree](#))
Internal ctor. Use static method CreateRoot<T>(T, BinTree<T>) to create a new tree, or SpawnChild(T, bool) to spawn a new node.
- [NodeControl CreateUIElem](#) (T value, bool bLeft)
Create a new UI element for a node and add it to the Canvas
- [NodeArrow CreateArrowTo](#) ([Node< T > node](#))
Create a UI arrow from this node pointing to another node, add it to the Canvas, and return it.

Static Private Member Functions

- static Point [GetLocOf](#) ([NodeControl](#) control)
Get location of a specified control.

8.6.1 Detailed Description

A binary tree's node.

Template Parameters

<i>T</i>	The type of data this node holds
----------	----------------------------------

Type Constraints

***T* : [IComparable< T >](#)**

8.6.2 Constructor & Destructor Documentation

8.6.2.1 Node()

```
BinTreeVisualization.Algorithms.Node< T >.Node (
    T value,
    BinTree< T > tree ) [private]
```

Internal ctor. Use static method `CreateRoot<T>(T, BinTree<T>)` to create a new tree, or `SpawnChild(T, bool)` to spawn a new node.

Parameters

<i>value</i>	Value for the new node to be spawned with
<i>tree</i>	The tree the node is in

8.6.3 Member Function Documentation

8.6.3.1 ActivateBlue()

```
void BinTreeVisualization.Algorithms.Node< T >.ActivateBlue (
    bool bRecursive = false )
```

Change node's fill color to blue with an animation.

Parameters

<i>bRecursive</i>	Whether to color the node itself, or the node and all its children as well
-------------------	--

8.6.3.2 AdoptChild()

```
Node< T > BinTreeVisualization.Algorithms.Node< T >.AdoptChild (
    Node< T > node )
```

Adopt a child node. The child along its entire subtree will be moved to the correct side of this node.

If selves children slots are full, the passed-in *node* will itself adopt one of the current children in order to create a chain.

Parameters

<i>node</i>	The node to adopt
-------------	-------------------

Returns

The inserted node

8.6.3.3 Blink()

```
void BinTreeVisualization.Algorithms.Node< T >.Blink (
    bool bRecursive = false )
```

Blink the node's UI control in blue.

Parameters

<i>bRecursive</i>	Whether to blink the node itself, or the node and all its children as well
-------------------	--

8.6.3.4 CalcHeight()

```
int BinTreeVisualization.Algorithms.Node< T >.CalcHeight ( )
```

Calculate node's height recursively.

Returns

Height of the node

8.6.3.5 CreateArrowTo()

```
NodeArrow BinTreeVisualization.Algorithms.Node< T >.CreateArrowTo (
    Node< T > node ) [private]
```

Create a UI arrow from this node pointing to another node, add it to the Canvas, and return it.

Parameters

<i>node</i>	The node the arrow should point to
-------------	------------------------------------

Returns

The created UI element.

8.6.3.6 CreateRoot< T >()

```
static Node< T > BinTreeVisualization.Algorithms.Node< T >.CreateRoot< T > (
    T value,
    BinTree< T > tree ) [static]
```

Create a root node to start a new tree.

Template Parameters

<i>T</i>	
----------	--

Parameters

<i>value</i>	Value to create the root with
<i>tree</i>	The tree in which the root should be created

Returns

The newly created Node<T>

Type Constraints

***T* : *Comparable*<*T*>**

8.6.3.7 CreateUIElem()

```
NodeControl BinTreeVisualization.Algorithms.Node< T >.CreateUIElem (
    T value,
    bool bLeft ) [private]
```

Create a new UI element for a node and add it to the Canvas

Parameters

<i>value</i>	Value of the new node
<i>bLeft</i>	Whether to prepare the child on parent's left side.

Returns

The created UI element.

8.6.3.8 DeactivateBlue()

```
void BinTreeVisualization.Algorithms.Node< T >.DeactivateBlue (
    bool bRecursive = false )
```

Reset node's fill color.

Parameters

<i>bRecursive</i>	Whether to blink the node itself, or the node and all its children as well
-------------------	--

8.6.3.9 GetDepth()

```
int BinTreeVisualization.Algorithms.Node< T >.GetDepth ( )
```

Get depth, i.e. how many levels deep is this node in the tree.

Returns

Depth of the node

8.6.3.10 GetHeight()

```
int BinTreeVisualization.Algorithms.Node< T >.GetHeight ( )
```

Get height of the node, i.e. how many levels deep is the deepest child of this node.

Returns

Height of the node

8.6.3.11 GetLocOf()

```
static Point BinTreeVisualization.Algorithms.Node< T >.GetLocOf (
    NodeControl control ) [static], [private]
```

Get location of a specified control.

Parameters

<i>control</i>	The control to get location of
----------------	--------------------------------

Returns

The location of the specified control

8.6.3.12 GetNodeBalance()

```
int BinTreeVisualization.Algorithms.Node< T >.GetNodeBalance ( )
```

Get the balance of the node - how much taller is its left children tree than the right one.

Returns

The node balance

8.6.3.13 GetRelativeNode()

```
Node< T > BinTreeVisualization.Algorithms.Node< T >.GetRelativeNode (
    int toRight )
```

Get the node at a relative position to the right from the current node, as if all nodes on this level were taken.

Returns null if the spot at this location is empty.

Parameters

<i>toRight</i>	Relative amount of node-slots to the right
----------------	--

Returns

The node at the requested relative position.

8.6.3.14 IsLeaf()

```
bool BinTreeVisualization.Algorithms.Node< T >.IsLeaf ( )
```

Whether the node is a leaf (i.e. has no children)

Returns

Whether the node is a leaf (i.e. has no children)

8.6.3.15 MoveByLoc()

```
void BinTreeVisualization.Algorithms.Node< T >.MoveByLoc (
    Point loc,
    bool bDelayAnimation = false )
```

Move the associated node control by a specified amount.

Parameters

<i>loc</i>	Final location to move the node to
<i>bDelayAnimation</i>	If <code>true</code> , cache the desired location and delay execution of the animation until <code>PlayDelayedAnimation</code> is called.

8.6.3.16 MoveChildrenByLoc()

```
void BinTreeVisualization.Algorithms.Node< T >.MoveChildrenByLoc (
    Point deltaLoc,
    bool bDelayAnimation = false )
```

Move all the children of this node (but NOT this node itself) by a delta location.

Parameters

<i>deltaLoc</i>	Delta location to move the tree by
<i>bDelayAnimation</i>	If <code>true</code> , cache the desired location and delay execution of the animation until <code>PlayDelayedAnimation</code> is called.

8.6.3.17 MoveToLoc()

```
void BinTreeVisualization.Algorithms.Node< T >.MoveToLoc (
    Point loc,
    bool bDelayAnimation = false )
```

Move the associated node control to specified location over 0.5 seconds.

Parameters

<i>loc</i>	Location to move to
<i>bDelayAnimation</i>	If <code>true</code> , cache the desired location and delay execution of the animation until <code>PlayDelayedAnimation</code> is called.

8.6.3.18 MoveToLocWithAnim()

```
void BinTreeVisualization.Algorithms.Node< T >.MoveToLocWithAnim (
    Point loc,
    bool bDelayAnimation = false )
```

Move the associated node control to specified location over 0.5 seconds. Always play the animation - regardless of instant mode or others.

Parameters

<i>loc</i>	Location to move to
<i>bDelayAnimation</i>	If <code>true</code> , cache the desired location and delay execution of the animation until <code>PlayDelayedAnimation</code> is called.

8.6.3.19 MoveTreeByLoc()

```
void BinTreeVisualization.Algorithms.Node< T >.MoveTreeByLoc (
    Point deltaLoc,
    bool bDelayAnimation = false )
```

Move the associated node control by a delta location and make its all its children follow it.

Parameters

<i>deltaLoc</i>	Delta location to move the tree by
<i>bDelayAnimation</i>	If <code>true</code> , cache the desired location and delay execution of the animation until <code>PlayDelayedAnimation</code> is called.

8.6.3.20 MoveTreeToLoc()

```
void BinTreeVisualization.Algorithms.Node< T >.MoveTreeToLoc (
    Point loc,
    bool bDelayAnimation = false )
```

Move the associated node control to a location and make its all its children follow it.

Parameters

<i>loc</i>	Final location to move the node to
<i>bDelayAnimation</i>	If <code>true</code> , cache the desired location and delay execution of the animation until <code>PlayDelayedAnimation</code> is called.

8.6.3.21 operator<()

```
static bool BinTreeVisualization.Algorithms.Node< T >.operator< (
    Node< T > self,
    Node< T > other ) [static]
```

Compare nodes value-wise.

Parameters

<i>self</i>	Left item to compare
<i>other</i>	Right item to compare

Returns

Whether the value of left is smaller than the value of right

8.6.3.22 operator<=()

```
static bool BinTreeVisualization.Algorithms.Node< T >.operator<= (
    Node< T > self,
    Node< T > other ) [static]
```

Compare nodes value-wise.

Parameters

<i>self</i>	Left item to compare
<i>other</i>	Right item to compare

Returns

Whether the value of left is smaller or equals to the value of right

8.6.3.23 operator>()

```
static bool BinTreeVisualization.Algorithms.Node< T >.operator> (
    Node< T > self,
    Node< T > other ) [static]
```

Compare nodes value-wise.

Parameters

<i>self</i>	Left item to compare
<i>other</i>	Right item to compare

Returns

Whether the value of left is bigger than the value of right

8.6.3.24 operator>=()

```
static bool BinTreeVisualization.Algorithms.Node< T >.operator>= (
    Node< T > self,
    Node< T > other ) [static]
```

Compare nodes value-wise.

Parameters

<i>self</i>	Left item to compare
<i>other</i>	Right item to compare

Returns

Whether the value of left is bigger or equals to the value of right

8.6.3.25 OrphanChildren()

```
void BinTreeVisualization.Algorithms.Node< T >.OrphanChildren (
    bool OrphanLeftChild = true,
    bool OrphanRightChild = true )
```

Orphan the children of this node.

Parameters

<i>OrphanLeftChild</i>	Whether to orphan the left child
<i>OrphanRightChild</i>	Whether to orphan the right child

8.6.3.26 SpawnChild()

```
Node< T > BinTreeVisualization.Algorithms.Node< T >.SpawnChild (
    T value,
    bool bLeft )
```

Spawn a new node as child of this node. This is the only way to spawn new nodes.

Parameters

<i>value</i>	Value for the new node to be spawned with
<i>bLeft</i>	Whether to spawn new node as this node's left child

Returns

The newly spawned node

8.6.3.27 SwapValues()

```
void BinTreeVisualization.Algorithms.Node< T >.SwapValues (
    Node< T > other )
```

Swap value of this node with value from another one. Does NOT ensure binary tree properties are maintained!

Parameters

<i>other</i>	The other node
--------------	----------------

8.6.3.28 Traverse()

```
IEnumerable< Node< T > > BinTreeVisualization.Algorithms.Node< T >.Traverse ( )
```

Traverse the tree in Level Order Traversal.

Returns

Returns IEnumerable that contains this and all children nodes

8.6.3.29 TraverseAncestors()

```
IEnumerable< Node< T > > BinTreeVisualization.Algorithms.Node< T >.TraverseAncestors ( )
```

Traverse ancestors of this node in the order parent -> grandparent -> ... -> root.

Returns

Returns IEnumerable that contains this and all parent nodes going from the bottom

8.6.4 Property Documentation**8.6.4.1 this[Side side]**

```
Node<T>? BinTreeVisualization.Algorithms.Node< T >.this[Side side] [get], [set]
```

Get or set a node by its side.

Parameters

<i>side</i>	The side
-------------	----------

Returns

Exceptions

<i>ArgumentOutOfRangeException</i>	
------------------------------------	--

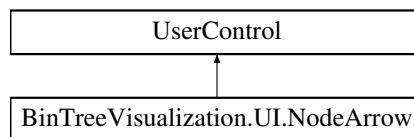
The documentation for this class was generated from the following files:

- S:/Uni/Algosity/BinTreeVisualization/Algorithms/Node.cs
- S:/Uni/Algosity/BinTreeVisualization/Algorithms/NodeUI.cs

8.7 BinTreeVisualization.UI.NodeArrow Class Reference

Represents an arrow that can point between nodes.

Inheritance diagram for BinTreeVisualization.UI.NodeArrow:



Public Member Functions

- void [RotateToTarget](#) (Point target)
Set the arrow to instantly rotate towards specified target.
- async void **RemoveSelf** ()
Remove this arrow from the canvas.
- void [MoveSourceToLoc](#) (Point newSource)
Repoint the source's location to new loc in a 0.5s animation.
- void [MoveTargetToLoc](#) (Point newTarget)
Repoint the target's location to new loc in a 0.5s animation.
- void [RotateToLoc](#) (Point newSource, Point newTarget)
Repoint the source and target locations to new locs in a 0.5s animation.
- void **TriggerSpawnAnim** ()
Trigger spawn fade-in animation.
- void **TriggerDespawnAnim** ()
Trigger despawn fade-out animation.

Static Public Member Functions

- static Point [GetUpperArrowSocket](#) (Point fromUpperLeftCorner)
Get location of a socket suitable to be pointed to by node's parent.
- static Point [GetLeftArrowSocket](#) (Point fromUpperLeftCorner)
Get location of a socket suitable for a parent to point to the left node.
- static Point [GetRightArrowSocket](#) (Point fromUpperLeftCorner)
Get location of socket suitable for a parent to point to the right node.
- static Point [GetNodeBorder](#) (Point fromUpperLeftCorner, double degrees)
Get location of node's border on the specified angle, assuming degrees = 0 is pointing to vec (1, 0)
- static Point [GetLowerArrowSocket](#) (Point fromUpperLeftCorner)
Get location of socket suitable for a parent to point right downwards.

Static Public Attributes

- static readonly DependencyProperty [SourceProp](#)
Current source dependency prop. Sets control's Canvas position automatically on Setter.
- static readonly DependencyProperty [TargetProp](#)
Current target dependency prop. Sets control's Canvas position automatically on Setter.

Properties

- Point **SelfLoc** [get]
Current in-UI location.
- Point **Target** [get, set]
The current target the arrow points to.
- Point **Source** [get, set]
The current source the arrow points from.
- Point **DesiredTarget** [get, private set]
The final target the arrow will point to after animations finish.
- Point **DesiredSource** [get, private set]
The final source the arrow will point from after animations finish.

Private Member Functions

- Canvas [GetCanvas](#) ()
Get the canvas this control is on.
- double [GetRotToTarget](#) (Point target)
*Get rotation in radians from current Source to the target .
Does NOT use DesiredSource as this method is being used in live calculations.*
- double [GetDistanceTo](#) (Point target)
*Get distance from current Source to the target .
Does NOT use DesiredSource as this method is being used in live calculations.*

Static Private Member Functions

- static Point [GetLocOf](#) ([NodeArrow](#) control)
Get location of a specified control.
- static void **OnTargetChanged** (DependencyObject d, DependencyPropertyChangedEventArgs e)
Refreshes the arrow's location on the Canvas.
- static void **OnSourceChanged** (DependencyObject d, DependencyPropertyChangedEventArgs e)
Refreshes the arrow's location on the Canvas.
- static double [GetGoodSideAngle](#) ()
*Get a good angle counting from node's side (1, 0) for an arrow to go out of considering the amount of free space between nodes.
Reads [Node<T>.ToBottomOffset](#) and [Node<T>.NodeHeight](#) for calculations.*

Private Attributes

- bool **LastSourceSideWasRight** = false

8.7.1 Detailed Description

Represents an arrow that can point between nodes.

8.7.2 Member Function Documentation

8.7.2.1 GetCanvas()

```
Canvas BinTreeVisualization.UI.NodeArrow.GetCanvas ( ) [private]
```

Get the canvas this control is on.

Returns

8.7.2.2 GetDistanceTo()

```
double BinTreeVisualization.UI.NodeArrow.GetDistanceTo (
    Point target ) [private]
```

Get distance from current Source to the *target* .

Does NOT use DesiredSource as this method is being used in live calculations.

Parameters

<i>target</i>	Absolute location of point to calculate the distance to
---------------	---

Returns

WPF length units from to the target

8.7.2.3 GetGoodSideAngle()

```
static double BinTreeVisualization.UI.NodeArrow.GetGoodSideAngle ( ) [static], [private]
```

Get a good angle counting from node's side (1, 0) for an arrow to go out of considering the amount of free space between nodes.

Reads [Node<T>.ToBottomOffset](#) and [Node<T>.NodeHeight](#) for calculations.

Returns

A visually pleasing angle.

8.7.2.4 GetLeftArrowSocket()

```
static Point BinTreeVisualization.UI.NodeArrow.GetLeftArrowSocket (
    Point fromUpperLeftCorner ) [static]
```

Get location of a socket suitable for a parent to point to the left node.

Parameters

<i>fromUpperLeftCorner</i>	Upper left coord of the parent for the arrow to be sourced from
----------------------------	---

Returns

Absolute position of the point on node's border

8.7.2.5 GetLocOf()

```
static Point BinTreeVisualization.UI.NodeArrow.GetLocOf (
    NodeArrow control ) [static], [private]
```

Get location of a specified control.

Parameters

<i>control</i>	The control to get location of
----------------	--------------------------------

Returns

The location of the specified control

8.7.2.6 GetLowerArrowSocket()

```
static Point BinTreeVisualization.UI.NodeArrow.GetLowerArrowSocket (
    Point fromUpperLeftCorner ) [static]
```

Get location of socket suitable for a parent to point right downwards.

Parameters

<i>fromUpperLeftCorner</i>	Upper left coord of the parent for the arrow to be sourced from
----------------------------	---

Returns

Absolute position of the point on node's border

8.7.2.7 GetNodeBorder()

```
static Point BinTreeVisualization.UI.NodeArrow.GetNodeBorder (
    Point fromUpperLeftCorner,
    double degrees ) [static]
```

Get location of node's border on the specified agle, assuming *degrees* = 0 is pointing to vec (1, 0)

Parameters

<i>fromUpperLeftCorner</i>	Upper left coord of the node
<i>degrees</i>	Degrees

Returns

Absolute position of the point on node's border

8.7.2.8 GetRightArrowSocket()

```
static Point BinTreeVisualization.UI.NodeArrow.GetRightArrowSocket (
    Point fromUpperLeftCorner ) [static]
```

Get location of socket suitable for a parent to point to the right node.

Parameters

<i>fromUpperLeftCorner</i>	Upper left coord of the parent for the arrow to be sourced from
----------------------------	---

Returns

Absolute position of the point on node's border

8.7.2.9 GetRotToTarget()

```
double BinTreeVisualization.UI.NodeArrow.GetRotToTarget (
    Point target ) [private]
```

Get rotation in radians from current Source to the *target* .

Does NOT use DesiredSource as this method is being used in live calculations.

Parameters

<i>target</i>	Absolute location of point to rotate to
---------------	---

Returns

Degrees from source to target

8.7.2.10 GetUpperArrowSocket()

```
static Point BinTreeVisualization.UI.NodeArrow.GetUpperArrowSocket (
    Point fromUpperLeftCorner ) [static]
```

Get location of a socket suitable to be pointed to by node's parent.

Parameters

<i>fromUpperLeftCorner</i>	Upper left coord of child node to point to
----------------------------	--

Returns

Absolute position of the point on node's border

8.7.2.11 MoveSourceToLoc()

```
void BinTreeVisualization.UI.NodeArrow.MoveSourceToLoc (
    Point newSource )
```

Repoint the source's location to new loc in a 0.5s animation.

Parameters

<i>newSource</i>	Upper left coord of the node's expected position
------------------	--

8.7.2.12 MoveTargetToLoc()

```
void BinTreeVisualization.UI.NodeArrow.MoveTargetToLoc (
    Point newTarget )
```


Repoint the target's location to new loc in a 0.5s animation.

Parameters

<i>newTarget</i>	Upper left coord of the node's expected position
------------------	--

8.7.2.13 RotateToLoc()

```
void BinTreeVisualization.UI.NodeArrow.RotateToLoc (
    Point newSource,
    Point newTarget )
```

Repoint the source and target locations to new locs in a 0.5s animation.

Parameters

<i>newSource</i>	The new source for the arrow to point from
<i>newTarget</i>	The new target for the arrow to point to

8.7.2.14 RotateToTarget()

```
void BinTreeVisualization.UI.NodeArrow.RotateToTarget (
    Point target )
```

Set the arrow to instantly rotate towards specified target.

Parameters

<i>target</i>	The target to rotate to
---------------	-------------------------

8.7.3 Member Data Documentation

8.7.3.1 SourceProp

```
readonly DependencyProperty BinTreeVisualization.UI.NodeArrow.SourceProp [static]
```

Initial value:

```
=
DependencyProperty.Register(
    "Source",
    typeof(Point),
    typeof(NodeArrow),
    new PropertyMetadata(new Point(0, 0), OnSourceChanged))
```

Current source dependency prop. Sets control's Canvas position automatically on Setter.

8.7.3.2 TargetProp

readonly DependencyProperty BinTreeVisualization.UI.NodeArrow.TargetProp [static]

Initial value:

```
=
DependencyProperty.Register(
    "Target",
    typeof(Point),
    typeof(NodeArrow),
    new PropertyMetadata(new Point(0, 0), OnTargetChanged))
```

Current target dependency prop. Sets control's Canvas position automatically on Setter.

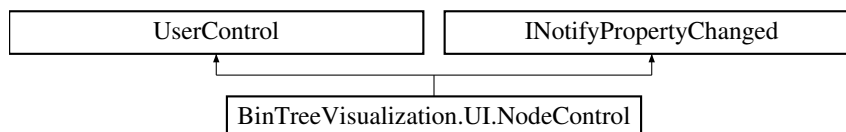
The documentation for this class was generated from the following file:

- S:/Uni/Algosity/BinTreeVisualization/UI/NodeArrow.xaml.cs

8.8 BinTreeVisualization.UI.NodeControl Class Reference

UI element that represents a node in a binary tree.

Inheritance diagram for BinTreeVisualization.UI.NodeControl:



Public Member Functions

- **NodeControl** (object value)
- void **Activate** ()
Highlight the node in the UI.
- void **Deactivate** ()
Remove highlight of the UI node.
- void **Blink** ()
Briefly blink the node's borders in blue.
- void **HighlightBlue** ()
Highlight the node in blue.
- void **DeactivateBlue** ()
Remove the blue highlight from the node.
- void **ActivateBlue** ()
Highlight the node in blue in the UI.
- void **MoveToLoc** (Point loc)
Move the associated node control to specified location over 0.5 seconds.

Protected Member Functions

- virtual void **OnPropertyChanged** ([CallerMemberName] string propertyName= "")

Properties

- Point **CurrLoc** [get]
- static Color **InactiveColor** [get]
- static Color **ActiveColor** [get]
- static Color **StrokeBase** [get]
- static Color **StrokeBlue** [get]
- static Color **RedColor** [get]
- object **Value** [get, set]
Node's value.
- string **ValToStr** [get]
Node's value as string. To be bound to the UI content.

Events

- PropertyChangedEventHandler **PropertyChanged**

Private Member Functions

- void **Node_OnClick** (object sender, MouseButtonEventArgs e)
*Click node to quickly select it in the parent UI.
Relies on the established structure of Main Window -> MainFrame -> BinTreeMan -> InputTextBox.*

8.8.1 Detailed Description

UI element that represents a node in a binary tree.

8.8.2 Member Function Documentation

8.8.2.1 MoveToLoc()

```
void BinTreeVisualization.UI.NodeControl.MoveToLoc (
    Point loc )
```

Move the associated node control to specified location over 0.5 seconds.

Parameters

<i>loc</i>	Location to move to
------------	---------------------

8.8.2.2 Node_OnClick()

```
void BinTreeVisualization.UI.NodeControl.Node_OnClick (
    object sender,
    MouseButtonEventArgs e ) [private]
```

Click node to quickly select it in the parent UI.

Relies on the established structure of Main Window -> MainFrame -> BinTreeMan -> InputTextBox.

Parameters

<i>sender</i>	
<i>e</i>	

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

- S:/Uni/Algosity/BinTreeVisualization/UI/NodeControl.xaml.cs

8.9 BinTreeVisualization.UI.PointExtensions Class Reference

Static Public Member Functions

- static double [Size](#) (this Vector p)
Gets the size of the vector.
- static Vector [Normal](#) (this Vector p)
Normalizes the vector.

8.9.1 Member Function Documentation

8.9.1.1 Normal()

```
static Vector BinTreeVisualization.UI.PointExtensions.Normal (
    this Vector p ) [static]
```

Normalizes the vector.

Parameters

<i>p</i>	The vector
----------	------------

Returns

A new vector, which is a normalized copy of the source

8.9.1.2 Size()

```
static double BinTreeVisualization.UI.PointExtensions.Size (
    this Vector p ) [static]
```

Gets the size of the vector.

Parameters

<i>p</i>	The vector
----------	------------

Returns

The size of the vector

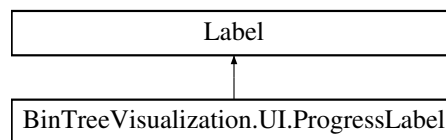
The documentation for this class was generated from the following file:

- S:/Uni/Algosity/BinTreeVisualization/UI/NodeArrow.xaml.cs

8.10 BinTreeVisualization.UI.ProgressLabel Class Reference

A label that can represent progress which supports spawn/despawn animations and colors.

Inheritance diagram for BinTreeVisualization.UI.ProgressLabel:

**Public Member Functions**

- void **TriggerSpawnAnim** ()
Trigger spawn fade-in animation.
- void **TriggerDespawnAnim** ()
Trigger despawn fade-out animation.

8.10.1 Detailed Description

A label that can represent progress which supports spawn/despawn animations and colors.

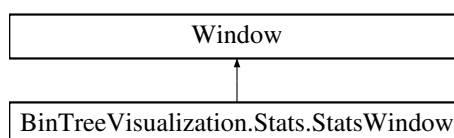
The documentation for this class was generated from the following file:

- S:/Uni/Algosity/BinTreeVisualization/UI/ProgressLabel.xaml.cs

8.11 BinTreeVisualization.Stats.StatsWindow Class Reference

Interaction logic for StatsWindow.xaml.

Inheritance diagram for BinTreeVisualization.Stats.StatsWindow:



Public Member Functions

- **StatsWindow** ([TreeStats](#) stats)

Properties

- [TreeStats](#) **StatsRef** [get, set]
Ref to the tree's data.

Private Member Functions

- void **OnDataChanged** ()
Function to call when the tree's data changes.
- void **OnClosed** (object sender, EventArgs e)
Function to call on window's closing. Unsubscribes from the tree's data changed event.

8.11.1 Detailed Description

Interaction logic for StatsWindow.xaml.

8.11.2 Member Function Documentation

8.11.2.1 OnClosed()

```
void BinTreeVisualization.Stats.StatsWindow.OnClosed (
    object sender,
    EventArgs e ) [private]
```

Function to call on window's closing. Unsubscribes from the tree's data changed event.

Parameters

<i>sender</i>	
<i>e</i>	

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

- S:/Uni/Algosy/BinTreeVisualization/Stats/StatsWindow.xaml.cs

8.12 BinTreeVisualization.Algorithms.TextActionColorsHelper Class Reference

Helper for colors of a TextAction

Static Public Member Functions

- static System.Windows.Media.Color [GetColor](#) (this [TextAction](#) textAction)
Convert value from a color enum to a System.Windows.Media.Color color.

8.12.1 Detailed Description

Helper for colors of a TextAction

8.12.2 Member Function Documentation

8.12.2.1 GetColor()

```
static System.Windows.Media.Color BinTreeVisualization.Algorithms.TextActionColorsHelper.GetColor (
    this TextAction textAction ) [static]
```

Convert value from a color enum to a System.Windows.Media.Color color.

Parameters

<i>textAction</i>	The text action to get the respective color of
-------------------	--

Returns

A System.Windows.Media.Color that corresponds to the passed TextAction

Exceptions

<i>NotImplementedException</i>	
--------------------------------	--

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

- S:/Uni/Algosy/BinTreeVisualization/Algorithms/BinTree.cs

8.13 BinTreeVisualization.Stats.TreeStats Class Reference

Holds statistics of operations performed on a binary tree.

Public Member Functions

- void [AddStats](#) ([OperationStats](#) stats, [OperationType](#) type)
Add stats performed during a specified operation.
- IEnumerable< [OperationStats](#) > [GetListWithAverages](#) ([OperationType](#) type)

Calculate the list of statistics, with the average amount of operations per the count of items in the tree.

- void **ShowWindow** ()
Show a window with plots holding the stats.
- void **Refresh** (StatsWindow window)
Refresh a plots window with the newest data.
- delegate void **StatsWindowEventHandler** ()
Event invoked when new stats are added.

Events

- **StatsWindowEventHandler OnDataChanged**
Event invoked when new stats are added.

Private Member Functions

- List< **OperationStats** > **GetList** (OperationType type)
Get the list of data associated with the specified operation type.
- void **FillPlot** (Plot plot, OperationType type)
Fill the plot with the data of the specified operation type.
- void **FillPlot** (WpfPlot plot, OperationType type)

Private Attributes

- List< **OperationStats** > **InsertStats** = []
Stats for insert operations.
- List< **OperationStats** > **DeleteStats** = []
Stats for delete operations.
- List< **OperationStats** > **SearchStats** = []
Stats for search operations.

8.13.1 Detailed Description

Holds statistics of operations performed on a binary tree.

8.13.2 Member Function Documentation

8.13.2.1 AddStats()

```
void BinTreeVisualization.Stats.TreeStats.AddStats (
    OperationStats stats,
    OperationType type )
```

Add stats performed during a specified operation.

Parameters

<i>stats</i>	The stats to add
<i>type</i>	The type of the operation

8.13.2.2 FillPlot()

```
void BinTreeVisualization.Stats.TreeStats.FillPlot (
    Plot plot,
    OperationType type ) [private]
```

Fill the plot with the data of the specified operation type.

Parameters

<i>plot</i>	The plot
<i>type</i>	The operaion type

8.13.2.3 GetList()

```
List< OperationStats > BinTreeVisualization.Stats.TreeStats.GetList (
    OperationType type ) [private]
```

Get the list of data associated with the specified operation type.

Parameters

<i>type</i>	The operation type
-------------	--------------------

Returns

List of data with the specified operation type

Exceptions

<i>ArgumentException</i>	
--------------------------	--

8.13.2.4 GetListWithAverages()

```
IEnumerable< OperationStats > BinTreeVisualization.Stats.TreeStats.GetListWithAverages (
    OperationType type )
```

Calculate the list of statistics, with the average amount of operations per the count of items in the tree.

Parameters

<i>type</i>	The operation type to get results for
-------------	---------------------------------------

Returns

An IEnumerable<T> with average comparisons and travels for each recorded tree size.

8.13.2.5 Refresh()

```
void BinTreeVisualization.Stats.TreeStats.Refresh (
    StatsWindow window )
```

Refresh a plots window with the newest data.

Parameters

<i>window</i>	The StatsWindow to refresh the data on
---------------	--

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

- S:/Uni/Algosity/BinTreeVisualization/Stats/TreeStats.cs

Index

- ActivateBlue
 - BinTreeVisualization.Algorithms.Node< T >, [42](#)
- AddRandomItems
 - BinTreeVisualization.UI.BinTreeMan, [34](#)
- AddStats
 - BinTreeVisualization.Stats.TreeStats, [65](#)
- AdoptChild
 - BinTreeVisualization.Algorithms.Node< T >, [42](#)
- AnimAdoption
 - BinTreeVisualization.Algorithms.BinTree< T >, [21](#)
- BalanceTreeIfNeeded
 - BinTreeVisualization.Algorithms.BinTree< T >, [21](#)
- Binary Tree Visualization, [5](#)
- BinTreeRow< T >
 - BinTreeVisualization.Algorithms, [14](#)
- BinTreeVisualization, [13](#)
- BinTreeVisualization.Algorithms, [13](#)
 - BinTreeRow< T >, [14](#)
- BinTreeVisualization.Algorithms.BinTree< T >, [17](#)
 - AnimAdoption, [21](#)
 - BalanceTreeIfNeeded, [21](#)
 - Blink, [21](#)
 - CreateRoot, [22](#)
 - Delay, [22](#)
 - Delete, [22](#), [23](#)
 - Find, [23](#)
 - FinishOperationStats, [24](#)
 - GetCanvas, [24](#)
 - GetHeight, [24](#)
 - GetMax, [24](#), [25](#)
 - GetMin, [25](#)
 - GetRow, [26](#)
 - GetRows, [26](#)
 - Insert, [26](#)
 - OperationGuard, [27](#)
 - Rotate, [27](#)
 - RotateLeft, [27](#)
 - RotateRight, [28](#)
 - SwapNodes, [28](#)
 - SwapValues, [28](#)
 - VerifyTreesNotEmpty, [29](#)
 - VerifyTreeLayout, [29](#)
- BinTreeVisualization.Algorithms.Node< T >, [38](#)
 - ActivateBlue, [42](#)
 - AdoptChild, [42](#)
 - Blink, [42](#)
 - CalcHeight, [43](#)
 - CreateArrowTo, [43](#)
 - CreateRoot< T >, [43](#)
 - CreateUIElem, [44](#)
 - DeactivateBlue, [44](#)
 - GetDepth, [44](#)
 - GetHeight, [45](#)
 - GetLocOf, [45](#)
 - GetNodeBalance, [45](#)
 - GetRelativeNode, [45](#)
 - IsLeaf, [46](#)
 - MoveByLoc, [46](#)
 - MoveChildrenByLoc, [46](#)
 - MoveToLoc, [46](#)
 - MoveToLocWithAnim, [47](#)
 - MoveTreeByLoc, [47](#)
 - MoveTreeToLoc, [47](#)
 - Node, [42](#)
 - operator<, [48](#)
 - operator<=, [48](#)
 - operator>, [48](#)
 - operator>=, [49](#)
 - OrphanChildren, [49](#)
 - SpawnChild, [49](#)
 - SwapValues, [50](#)
 - this[Side side], [50](#)
 - Traverse, [50](#)
 - TraverseAncestors, [50](#)
- BinTreeVisualization.Algorithms.TextActionColorsHelper,
 - [63](#)
 - GetColor, [64](#)
- BinTreeVisualization.App, [17](#)
- BinTreeVisualization.MainWindow, [38](#)
- BinTreeVisualization.Stats, [14](#)
 - OperationStats, [15](#)
- BinTreeVisualization.Stats.StatsWindow, [62](#)
 - OnClosed, [63](#)
- BinTreeVisualization.Stats.TreeStats, [64](#)
 - AddStats, [65](#)
 - FillPlot, [66](#)
 - GetList, [66](#)
 - GetListWithAverages, [66](#)
 - Refresh, [66](#)
- BinTreeVisualization.UI, [15](#)
- BinTreeVisualization.UI.BinTreeControl, [29](#)
 - GetNewScale< T >, [30](#)
 - GetNodesTotalWidth< T >, [31](#)
 - LayoutTreeNSquare< T >, [31](#)
 - ResetText, [32](#)
 - SetScaleAnim, [32](#)
 - SetText, [32](#)
 - VerifyScale< T >, [32](#)

- BinTreeVisualization.UI.BinTreeMan, 33
 - AddRandomItems, 34
 - GetDoubleArg, 35
 - OnDelete, 35
 - OnFind, 35
 - OnGetMax, 35
 - OnGetMin, 36
 - OnInsert, 36
 - OnInsertManyItems, 36
 - OnKeyDownBackground, 37
 - OnKeyDownTextbox, 37
 - OnOpenStatsWindow, 37
 - SwitchAddItemsCount, 37
- BinTreeVisualization.UI.NodeArrow, 51
 - GetCanvas, 53
 - GetDistanceTo, 53
 - GetGoodSideAngle, 54
 - GetLeftArrowSocket, 54
 - GetLocOf, 54
 - GetLowerArrowSocket, 54
 - GetNodeBorder, 55
 - GetRightArrowSocket, 55
 - GetRotToTarget, 55
 - GetUpperArrowSocket, 56
 - MoveSourceToLoc, 56
 - MoveTargetToLoc, 56
 - RotateToLoc, 57
 - RotateToTarget, 57
 - SourceProp, 57
 - TargetProp, 57
- BinTreeVisualization.UI.NodeControl, 58
 - MoveToLoc, 59
 - Node_OnClick, 59
- BinTreeVisualization.UI.PointExtensions, 61
 - Normal, 61
 - Size, 61
- BinTreeVisualization.UI.ProgressLabel, 62
- Blink
 - BinTreeVisualization.Algorithms.BinTree< T >, 21
 - BinTreeVisualization.Algorithms.Node< T >, 42
- CalcHeight
 - BinTreeVisualization.Algorithms.Node< T >, 43
- CreateArrowTo
 - BinTreeVisualization.Algorithms.Node< T >, 43
- CreateRoot
 - BinTreeVisualization.Algorithms.BinTree< T >, 22
- CreateRoot< T >
 - BinTreeVisualization.Algorithms.Node< T >, 43
- CreateUIElem
 - BinTreeVisualization.Algorithms.Node< T >, 44
- DeactivateBlue
 - BinTreeVisualization.Algorithms.Node< T >, 44
- Delay
 - BinTreeVisualization.Algorithms.BinTree< T >, 22
- Delete
 - BinTreeVisualization.Algorithms.BinTree< T >, 22, 23
- FillPlot
 - BinTreeVisualization.Stats.TreeStats, 66
- Find
 - BinTreeVisualization.Algorithms.BinTree< T >, 23
- FinishOperationStats
 - BinTreeVisualization.Algorithms.BinTree< T >, 24
- General assignment, 1
- GetCanvas
 - BinTreeVisualization.Algorithms.BinTree< T >, 24
 - BinTreeVisualization.UI.NodeArrow, 53
- GetColor
 - BinTreeVisualization.Algorithms.TextActionColorsHelper, 64
- GetDepth
 - BinTreeVisualization.Algorithms.Node< T >, 44
- GetDistanceTo
 - BinTreeVisualization.UI.NodeArrow, 53
- GetDoubleArg
 - BinTreeVisualization.UI.BinTreeMan, 35
- GetGoodSideAngle
 - BinTreeVisualization.UI.NodeArrow, 54
- GetHeight
 - BinTreeVisualization.Algorithms.BinTree< T >, 24
 - BinTreeVisualization.Algorithms.Node< T >, 45
- GetLeftArrowSocket
 - BinTreeVisualization.UI.NodeArrow, 54
- GetList
 - BinTreeVisualization.Stats.TreeStats, 66
- GetListWithAverages
 - BinTreeVisualization.Stats.TreeStats, 66
- GetLocOf
 - BinTreeVisualization.Algorithms.Node< T >, 45
 - BinTreeVisualization.UI.NodeArrow, 54
- GetLowerArrowSocket
 - BinTreeVisualization.UI.NodeArrow, 54
- GetMax
 - BinTreeVisualization.Algorithms.BinTree< T >, 24, 25
- GetMin
 - BinTreeVisualization.Algorithms.BinTree< T >, 25
- GetNewScale< T >
 - BinTreeVisualization.UI.BinTreeControl, 30
- GetNodeBalance
 - BinTreeVisualization.Algorithms.Node< T >, 45
- GetNodeBorder
 - BinTreeVisualization.UI.NodeArrow, 55
- GetNodesTotalWidth< T >
 - BinTreeVisualization.UI.BinTreeControl, 31
- GetRelativeNode
 - BinTreeVisualization.Algorithms.Node< T >, 45
- GetRightArrowSocket
 - BinTreeVisualization.UI.NodeArrow, 55
- GetRotToTarget
 - BinTreeVisualization.UI.NodeArrow, 55
- GetRow
 - BinTreeVisualization.Algorithms.BinTree< T >, 26
- GetRows
 - BinTreeVisualization.Algorithms.BinTree< T >, 26

- GetUpperArrowSocket
 - BinTreeVisualization.UI.NodeArrow, [56](#)
- Insert
 - BinTreeVisualization.Algorithms.BinTree< T >, [26](#)
- IsLeaf
 - BinTreeVisualization.Algorithms.Node< T >, [46](#)
- LayoutTreeNSquare< T >
 - BinTreeVisualization.UI.BinTreeControl, [31](#)
- MoveByLoc
 - BinTreeVisualization.Algorithms.Node< T >, [46](#)
- MoveChildrenByLoc
 - BinTreeVisualization.Algorithms.Node< T >, [46](#)
- MoveSourceToLoc
 - BinTreeVisualization.UI.NodeArrow, [56](#)
- MoveTargetToLoc
 - BinTreeVisualization.UI.NodeArrow, [56](#)
- MoveToLoc
 - BinTreeVisualization.Algorithms.Node< T >, [46](#)
 - BinTreeVisualization.UI.NodeControl, [59](#)
- MoveToLocWithAnim
 - BinTreeVisualization.Algorithms.Node< T >, [47](#)
- MoveTreeByLoc
 - BinTreeVisualization.Algorithms.Node< T >, [47](#)
- MoveTreeToLoc
 - BinTreeVisualization.Algorithms.Node< T >, [47](#)
- Node
 - BinTreeVisualization.Algorithms.Node< T >, [42](#)
- Node_OnClick
 - BinTreeVisualization.UI.NodeControl, [59](#)
- Normal
 - BinTreeVisualization.UI.PointExtensions, [61](#)
- OnClosed
 - BinTreeVisualization.Stats.StatsWindow, [63](#)
- OnDelete
 - BinTreeVisualization.UI.BinTreeMan, [35](#)
- OnFind
 - BinTreeVisualization.UI.BinTreeMan, [35](#)
- OnGetMax
 - BinTreeVisualization.UI.BinTreeMan, [35](#)
- OnGetMin
 - BinTreeVisualization.UI.BinTreeMan, [36](#)
- OnInsert
 - BinTreeVisualization.UI.BinTreeMan, [36](#)
- OnInsertManyItems
 - BinTreeVisualization.UI.BinTreeMan, [36](#)
- OnKeyDownBackground
 - BinTreeVisualization.UI.BinTreeMan, [37](#)
- OnKeyDownTextbox
 - BinTreeVisualization.UI.BinTreeMan, [37](#)
- OnOpenStatsWindow
 - BinTreeVisualization.UI.BinTreeMan, [37](#)
- OperationGuard
 - BinTreeVisualization.Algorithms.BinTree< T >, [27](#)
- OperationStats
 - BinTreeVisualization.Stats, [15](#)
- operator<
 - BinTreeVisualization.Algorithms.Node< T >, [48](#)
- operator<=
 - BinTreeVisualization.Algorithms.Node< T >, [48](#)
- operator>
 - BinTreeVisualization.Algorithms.Node< T >, [48](#)
- operator>=
 - BinTreeVisualization.Algorithms.Node< T >, [49](#)
- OrphanChildren
 - BinTreeVisualization.Algorithms.Node< T >, [49](#)
- Project overview, [3](#)
- Refresh
 - BinTreeVisualization.Stats.TreeStats, [66](#)
- ResetText
 - BinTreeVisualization.UI.BinTreeControl, [32](#)
- Rotate
 - BinTreeVisualization.Algorithms.BinTree< T >, [27](#)
- RotateLeft
 - BinTreeVisualization.Algorithms.BinTree< T >, [27](#)
- RotateRight
 - BinTreeVisualization.Algorithms.BinTree< T >, [28](#)
- RotateToLoc
 - BinTreeVisualization.UI.NodeArrow, [57](#)
- RotateToTarget
 - BinTreeVisualization.UI.NodeArrow, [57](#)
- SetScaleAnim
 - BinTreeVisualization.UI.BinTreeControl, [32](#)
- SetText
 - BinTreeVisualization.UI.BinTreeControl, [32](#)
- Size
 - BinTreeVisualization.UI.PointExtensions, [61](#)
- SourceProp
 - BinTreeVisualization.UI.NodeArrow, [57](#)
- SpawnChild
 - BinTreeVisualization.Algorithms.Node< T >, [49](#)
- SwapNodes
 - BinTreeVisualization.Algorithms.BinTree< T >, [28](#)
- SwapValues
 - BinTreeVisualization.Algorithms.BinTree< T >, [28](#)
 - BinTreeVisualization.Algorithms.Node< T >, [50](#)
- SwitchAddItemsCount
 - BinTreeVisualization.UI.BinTreeMan, [37](#)
- TargetProp
 - BinTreeVisualization.UI.NodeArrow, [57](#)
- this[Side side]
 - BinTreeVisualization.Algorithms.Node< T >, [50](#)
- Traverse
 - BinTreeVisualization.Algorithms.Node< T >, [50](#)
- TraverseAncestors
 - BinTreeVisualization.Algorithms.Node< T >, [50](#)
- VerifyScale< T >
 - BinTreeVisualization.UI.BinTreeControl, [32](#)
- VerifyTreelsNotEmpty

BinTreeVisualization.Algorithms.BinTree< T >, [29](#)
VerifyTreeLayout
 BinTreeVisualization.Algorithms.BinTree< T >, [29](#)