

CS162 - Algorithm Visualizer

Generated by Doxygen 1.9.6



---

<b>1 Deprecated List</b>	<b>1</b>
<b>2 Namespace Index</b>	<b>3</b>
2.1 Namespace List . . . . .	3
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy . . . . .	5
<b>4 Class Index</b>	<b>7</b>
4.1 Class List . . . . .	7
<b>5 File Index</b>	<b>11</b>
5.1 File List . . . . .	11
<b>6 Namespace Documentation</b>	<b>15</b>
6.1 Algorithm Namespace Reference . . . . .	15
6.2 Animation Namespace Reference . . . . .	15
6.3 AnimationFactory Namespace Reference . . . . .	15
6.3.1 Detailed Description . . . . .	16
6.3.2 Function Documentation . . . . .	16
6.3.2.1 BlendColor() . . . . .	16
6.3.2.2 BounceOut() . . . . .	16
6.3.2.3 Dist() . . . . .	16
6.3.2.4 DrawActiveArrow() . . . . .	17
6.3.2.5 DrawCircularArrow() . . . . .	17
6.3.2.6 DrawDirectionalArrow() . . . . .	17
6.3.2.7 DrawDoubleActiveArrow() . . . . .	17
6.3.2.8 DrawDoubleDirectionalArrow() . . . . .	17
6.3.2.9 ElasticOut() . . . . .	18
6.3.2.10 InverseVector() . . . . .	18
6.3.2.11 MoveNode() . . . . .	18
6.3.2.12 ReCalculateEnds() . . . . .	18
6.4 Category Namespace Reference . . . . .	18
6.4.1 Enumeration Type Documentation . . . . .	18
6.4.1.1 ID . . . . .	18
6.5 ColorTheme Namespace Reference . . . . .	19
6.5.1 Enumeration Type Documentation . . . . .	19
6.5.1.1 ID . . . . .	19
6.6 Core Namespace Reference . . . . .	21
6.7 DataStructures Namespace Reference . . . . .	21
6.7.1 Enumeration Type Documentation . . . . .	22
6.7.1.1 ID . . . . .	22
6.8 Fonts Namespace Reference . . . . .	22
6.8.1 Enumeration Type Documentation . . . . .	22

---

6.8.1.1 ID . . . . .	22
6.9 global Namespace Reference . . . . .	23
6.9.1 Variable Documentation . . . . .	23
6.9.1.1 favicon . . . . .	23
6.9.1.2 kTitle . . . . .	23
6.9.1.3 SCREEN_HEIGHT . . . . .	23
6.9.1.4 SCREEN_WIDTH . . . . .	23
6.10 GUI Namespace Reference . . . . .	23
6.11 GUIComponent Namespace Reference . . . . .	24
6.12 GUIVisualizer Namespace Reference . . . . .	24
6.13 State Namespace Reference . . . . .	25
6.14 States Namespace Reference . . . . .	25
6.14.1 Enumeration Type Documentation . . . . .	25
6.14.1.1 ID . . . . .	25
6.15 Textures Namespace Reference . . . . .	26
6.15.1 Enumeration Type Documentation . . . . .	26
6.15.1.1 ID . . . . .	26
6.16 Utils Namespace Reference . . . . .	27
6.16.1 Function Documentation . . . . .	27
6.16.1.1 DrawIcon() . . . . .	27
6.16.1.2 DrawTextBoxed() . . . . .	27
6.16.1.3 DrawTextBoxedSelectable() . . . . .	28
6.16.1.4 OpenFileDialog() . . . . .	28
6.16.1.5 Rand() . . . . .	28
6.16.1.6 ReadInputFromFile() . . . . .	28
<b>7 Class Documentation</b> . . . . .	<b>29</b>
7.1 Algorithm::Algorithm< GUIAlgorithm, AnimationState > Class Template Reference . . . . .	29
7.1.1 Detailed Description . . . . .	30
7.1.2 Constructor & Destructor Documentation . . . . .	30
7.1.2.1 Algorithm() [1/2] . . . . .	31
7.1.2.2 Algorithm() [2/2] . . . . .	31
7.1.2.3 ~Algorithm() . . . . .	31
7.1.3 Member Function Documentation . . . . .	31
7.1.3.1 ApplyInput() . . . . .	31
7.1.3.2 Empty() . . . . .	32
7.1.3.3 EmptyGenerator() . . . . .	32
7.1.3.4 GenerateAnimation() . . . . .	32
7.1.3.5 GenerateRelayoutAnimation() . . . . .	33
7.1.3.6 InitAction() . . . . .	33
7.1.3.7 Random() . . . . .	33
7.1.3.8 RandomFixedSize() . . . . .	33

---

7.1.3.9 RandomFixedSizeGenerator()	34
7.1.3.10 RandomGenerator()	34
7.1.3.11 ReadFromExternalFile()	34
7.1.3.12 ReadFromFileGenerator()	35
7.1.3.13 UserDefined()	35
7.1.3.14 UserDefinedGenerator()	35
7.1.4 Member Data Documentation	36
7.1.4.1 animController	36
7.1.4.2 codeHighlighter	36
7.1.4.3 visualizer	36
7.2 Animation::AnimationController< T > Class Template Reference	37
7.2.1 Detailed Description	38
7.2.2 Member Typedef Documentation	39
7.2.2.1 Ptr	39
7.2.3 Constructor & Destructor Documentation	39
7.2.3.1 AnimationController()	39
7.2.3.2 ~AnimationController()	39
7.2.4 Member Function Documentation	39
7.2.4.1 AddAnimation()	39
7.2.4.2 Clear()	40
7.2.4.3 Continue()	40
7.2.4.4 CurrentAnimationIndex()	40
7.2.4.5 Done()	40
7.2.4.6 GetAnimateFrame()	40
7.2.4.7 GetAnimation()	41
7.2.4.8 GetAnimationDuration()	41
7.2.4.9 GetAnimationIndex()	41
7.2.4.10 GetNumAnimation()	42
7.2.4.11 GetSpeed()	42
7.2.4.12 GetStopDuration()	42
7.2.4.13 InteractionAllow()	42
7.2.4.14 InteractionLock()	43
7.2.4.15 IsInteractionAllow()	43
7.2.4.16 IsPlaying()	43
7.2.4.17 Pause()	43
7.2.4.18 PopAnimation()	43
7.2.4.19 Reset()	44
7.2.4.20 ResetCurrent()	44
7.2.4.21 RunAll()	44
7.2.4.22 SetAnimation()	44
7.2.4.23 SetSpeed()	44
7.2.4.24 StepBackward()	45

7.2.4.25 StepForward()	45
7.2.4.26 Update()	45
7.2.5 Member Data Documentation	46
7.2.5.1 animationGroup	46
7.2.5.2 defaultSpeed	46
7.2.5.3 mCurrentAnimationIndex	46
7.2.5.4 mCurrStopDuration	46
7.2.5.5 mInteractionLock	46
7.2.5.6 mPlaying	47
7.2.5.7 mSpeed	47
7.2.5.8 stopDuration	47
7.3 Animation::AnimationState< T > Class Template Reference	47
7.3.1 Detailed Description	49
7.3.2 Constructor & Destructor Documentation	49
7.3.2.1 AnimationState()	49
7.3.2.2 ~AnimationState()	49
7.3.3 Member Function Documentation	49
7.3.3.1 Done()	49
7.3.3.2 Draw()	49
7.3.3.3 GetActionDescription()	50
7.3.3.4 GetCurrentPlayingAt()	50
7.3.3.5 GetDataStructure()	50
7.3.3.6 GetDuration()	51
7.3.3.7 GetHighlightedLine()	51
7.3.3.8 PlayingAt()	51
7.3.3.9 Reset()	52
7.3.3.10 SetActionDescription()	52
7.3.3.11 SetAnimation()	52
7.3.3.12 SetDuration()	52
7.3.3.13 SetHighlightLine()	53
7.3.3.14 SetSourceDataStructure()	53
7.3.3.15 Update()	53
7.3.4 Member Data Documentation	53
7.3.4.1 actionDescription	54
7.3.4.2 mAnimation	54
7.3.4.3 mCurrentPlayingAt	54
7.3.4.4 mDataStructureBefore	54
7.3.4.5 mDuration	54
7.3.4.6 mHighlightedLine	55
7.4 Application Class Reference	55
7.4.1 Detailed Description	56
7.4.2 Constructor & Destructor Documentation	56

---

7.4.2.1 Application()	56
7.4.2.2 ~Application()	56
7.4.3 Member Function Documentation	56
7.4.3.1 Close()	56
7.4.3.2 Init()	57
7.4.3.3 LoadResources()	57
7.4.3.4 RegisterStates()	57
7.4.3.5 Render()	57
7.4.3.6 Run()	57
7.4.3.7 Update()	57
7.4.3.8 WindowClosed()	58
7.4.4 Member Data Documentation	58
7.4.4.1 categories	58
7.4.4.2 closed	58
7.4.4.3 dsInfo	59
7.4.4.4 fonts	59
7.4.4.5 images	59
7.4.4.6 mStack	59
7.5 State::ArrayState< T > Class Template Reference	60
7.5.1 Detailed Description	62
7.5.2 Member Typedef Documentation	62
7.5.2.1 Ptr	62
7.5.3 Constructor & Destructor Documentation	63
7.5.3.1 ArrayState()	63
7.5.3.2 ~ArrayState()	63
7.5.4 Member Function Documentation	63
7.5.4.1 AddAccessOperation()	63
7.5.4.2 AddDeleteOperation()	64
7.5.4.3 AddInitializeOperation()	64
7.5.4.4 AddInsertOperation()	64
7.5.4.5 AddIntFieldOperationOption()	64
7.5.4.6 AddNoFieldOperationOption()	64
7.5.4.7 AddOperations()	65
7.5.4.8 AddSearchOperation()	65
7.5.4.9 AddStringFieldOption()	65
7.5.4.10 AddUpdateOperation()	65
7.5.4.11 ClearAction()	66
7.5.4.12 ClearError()	66
7.5.4.13 Draw()	66
7.5.4.14 DrawCurrentActionText()	66
7.5.4.15 DrawCurrentErrorText()	66
7.5.4.16 GetContext()	67

7.5.4.17 InitNavigationBar() . . . . .	67
7.5.4.18 RequestStackClear() . . . . .	67
7.5.4.19 RequestStackPop() . . . . .	67
7.5.4.20 RequestStackPush() . . . . .	67
7.5.4.21 SetCurrentAction() . . . . .	68
7.5.4.22 SetCurrentError() . . . . .	68
7.5.4.23 Success() . . . . .	68
7.5.4.24 Update() . . . . .	69
7.5.5 Member Data Documentation . . . . .	69
7.5.5.1 activeDS . . . . .	69
7.5.5.2 animController . . . . .	69
7.5.5.3 codeHighlighter . . . . .	69
7.5.5.4 footer . . . . .	70
7.5.5.5 mContext . . . . .	70
7.5.5.6 mCurrentAction . . . . .	70
7.5.5.7 mCurrentError . . . . .	70
7.5.5.8 mStack . . . . .	70
7.5.5.9 navigation . . . . .	70
7.5.5.10 operationList . . . . .	71
7.6 GUIComponent::Button Class Reference . . . . .	71
7.6.1 Detailed Description . . . . .	74
7.6.2 Member Typedef Documentation . . . . .	74
7.6.2.1 Ptr . . . . .	74
7.6.3 Member Enumeration Documentation . . . . .	74
7.6.3.1 TextAlign . . . . .	74
7.6.4 Constructor & Destructor Documentation . . . . .	75
7.6.4.1 Button() [1/2] . . . . .	75
7.6.4.2 Button() [2/2] . . . . .	75
7.6.4.3 ~Button() . . . . .	75
7.6.5 Member Function Documentation . . . . .	75
7.6.5.1 deselect() . . . . .	75
7.6.5.2 DisableFitContent() . . . . .	76
7.6.5.3 Draw() . . . . .	76
7.6.5.4 DrawButtonText() . . . . .	77
7.6.5.5 EnableFitContent() . . . . .	77
7.6.5.6 FitContent() . . . . .	77
7.6.5.7 GetContentPos() . . . . .	77
7.6.5.8 GetContentSize() . . . . .	77
7.6.5.9 GetFontSize() . . . . .	77
7.6.5.10 GetHoverStatus() [1/2] . . . . .	77
7.6.5.11 GetHoverStatus() [2/2] . . . . .	78
7.6.5.12 GetPosition() . . . . .	78

---

7.6.5.13 GetSize()	78
7.6.5.14 GetVisible()	79
7.6.5.15 IsClicked()	79
7.6.5.16 IsSelectable()	79
7.6.5.17 IsSelected()	79
7.6.5.18 select()	80
7.6.5.19 SetAction()	80
7.6.5.20 SetActionOnHover()	80
7.6.5.21 SetButtonColor()	80
7.6.5.22 SetButtonHoverColor()	80
7.6.5.23 SetFontSize()	80
7.6.5.24 SetPosition() [1/2]	80
7.6.5.25 SetPosition() [2/2]	81
7.6.5.26 SetSize()	81
7.6.5.27 SetText()	81
7.6.5.28 SetTextAlignment()	81
7.6.5.29 SetTextColor()	82
7.6.5.30 SetVisible()	82
7.6.5.31 ToggleVisible()	82
7.6.5.32 UpdateMouseCursorWhenHover() [1/2]	82
7.6.5.33 UpdateMouseCursorWhenHover() [2/2]	83
7.6.6 Member Data Documentation	83
7.6.6.1 action	83
7.6.6.2 alignment	83
7.6.6.3 bound	83
7.6.6.4 buttonColorTheme	83
7.6.6.5 buttonHoverColorTheme	83
7.6.6.6 content	84
7.6.6.7 fitContent	84
7.6.6.8 fonts	84
7.6.6.9 fontSize	84
7.6.6.10 isHover	84
7.6.6.11 mActionOnHover	84
7.6.6.12 mIsSelected	84
7.6.6.13 mPosition	85
7.6.6.14 mVisible	85
7.6.6.15 textColorTheme	85
7.7 GUIComponent::Card Class Reference	85
7.7.1 Detailed Description	88
7.7.2 Member Typedef Documentation	88
7.7.2.1 Ptr	88
7.7.3 Constructor & Destructor Documentation	88

7.7.3.1 Card() [1/2] . . . . .	88
7.7.3.2 Card() [2/2] . . . . .	89
7.7.3.3 ~Card() . . . . .	89
7.7.4 Member Function Documentation . . . . .	89
7.7.4.1 deselect() . . . . .	89
7.7.4.2 Draw() . . . . .	89
7.7.4.3 DrawImage() . . . . .	89
7.7.4.4 DrawTitle() . . . . .	90
7.7.4.5 GetHoverStatus() [1/2] . . . . .	90
7.7.4.6 GetHoverStatus() [2/2] . . . . .	90
7.7.4.7 GetPosition() . . . . .	90
7.7.4.8 GetSize() . . . . .	91
7.7.4.9 GetVisible() . . . . .	91
7.7.4.10 isSelectable() . . . . .	91
7.7.4.11 isSelected() . . . . .	92
7.7.4.12 select() . . . . .	92
7.7.4.13 SetLink() . . . . .	92
7.7.4.14 SetPosition() [1/2] . . . . .	92
7.7.4.15 SetPosition() [2/2] . . . . .	93
7.7.4.16 SetStateID() . . . . .	93
7.7.4.17SetText() . . . . .	93
7.7.4.18 SetVisible() . . . . .	93
7.7.4.19 ToggleVisible() . . . . .	94
7.7.4.20 UpdateMouseCursorWhenHover() [1/2] . . . . .	94
7.7.4.21 UpdateMouseCursorWhenHover() [2/2] . . . . .	94
7.7.5 Member Data Documentation . . . . .	94
7.7.5.1 fonts . . . . .	94
7.7.5.2 hoverBounds . . . . .	95
7.7.5.3 isHover . . . . .	95
7.7.5.4 mIsSelected . . . . .	95
7.7.5.5 mPosition . . . . .	95
7.7.5.6 mVisible . . . . .	95
7.7.5.7 stateID . . . . .	95
7.7.5.8 thumbnail . . . . .	96
7.7.5.9 title . . . . .	96
7.7.5.10 toLink . . . . .	96
7.8 CategoryInfo Class Reference . . . . .	96
7.8.1 Detailed Description . . . . .	97
7.8.2 Member Function Documentation . . . . .	97
7.8.2.1 Get() . . . . .	97
7.8.2.2 Register() . . . . .	98
7.8.3 Member Data Documentation . . . . .	98

---

7.8.3.1 mFactories . . . . .	98
7.9 Algorithm::CircularLinkedList Class Reference . . . . .	99
7.9.1 Detailed Description . . . . .	101
7.9.2 Constructor & Destructor Documentation . . . . .	102
7.9.2.1 CircularLinkedList() [1/2] . . . . .	102
7.9.2.2 CircularLinkedList() [2/2] . . . . .	102
7.9.2.3 ~CircularLinkedList() . . . . .	102
7.9.3 Member Function Documentation . . . . .	102
7.9.3.1 ApplyInput() . . . . .	102
7.9.3.2 DeleteHead() . . . . .	103
7.9.3.3 DeleteMiddle() . . . . .	103
7.9.3.4 DeleteTail() . . . . .	103
7.9.3.5 Empty() . . . . .	103
7.9.3.6 EmptyGenerator() . . . . .	104
7.9.3.7 GenerateAnimation() . . . . .	104
7.9.3.8 GenerateRelayoutAnimation() . . . . .	104
7.9.3.9 HighlightArrowFromCur() . . . . .	105
7.9.3.10 HighlightCircularArrow() . . . . .	105
7.9.3.11 InitAction() . . . . .	105
7.9.3.12 InsertAfterTail() . . . . .	106
7.9.3.13 InsertHead() . . . . .	106
7.9.3.14 InsertMiddle() . . . . .	106
7.9.3.15 Random() . . . . .	107
7.9.3.16 RandomFixedSize() . . . . .	107
7.9.3.17 RandomFixedSizeGenerator() . . . . .	107
7.9.3.18 RandomGenerator() . . . . .	108
7.9.3.19 ReadFromExternalFile() . . . . .	108
7.9.3.20 ReadFromFileGenerator() . . . . .	108
7.9.3.21 ResetVisualizer() . . . . .	108
7.9.3.22 Search() . . . . .	109
7.9.3.23 size() . . . . .	109
7.9.3.24 Update() . . . . .	109
7.9.3.25 UserDefined() . . . . .	109
7.9.3.26 UserDefinedGenerator() . . . . .	110
7.9.4 Member Data Documentation . . . . .	110
7.9.4.1 animController . . . . .	110
7.9.4.2 codeHighlighter . . . . .	110
7.9.4.3 maxN . . . . .	111
7.9.4.4 visualizer . . . . .	111
7.10 GUIVisualizer::CircularLinkedList Class Reference . . . . .	111
7.10.1 Detailed Description . . . . .	115
7.10.2 Member Typedef Documentation . . . . .	115

---

7.10.2.1 Ptr . . . . .	115
7.10.3 Member Enumeration Documentation . . . . .	115
7.10.3.1 ArrowType . . . . .	115
7.10.3.2 Orientation . . . . .	116
7.10.4 Constructor & Destructor Documentation . . . . .	116
7.10.4.1 CircularLinkedList() [1/2] . . . . .	116
7.10.4.2 CircularLinkedList() [2/2] . . . . .	116
7.10.4.3 ~CircularLinkedList() . . . . .	117
7.10.5 Member Function Documentation . . . . .	117
7.10.5.1 DeleteNode() . . . . .	117
7.10.5.2 deselect() . . . . .	117
7.10.5.3 Draw() [1/2] . . . . .	117
7.10.5.4 Draw() [2/2] . . . . .	118
7.10.5.5 DrawArrow() . . . . .	118
7.10.5.6 DrawCurrent() . . . . .	118
7.10.5.7 GenerateNode() . . . . .	119
7.10.5.8 GetArrowType() . . . . .	119
7.10.5.9 GetChildren() . . . . .	119
7.10.5.10 GetCircularArrowType() . . . . .	119
7.10.5.11 GetCircularEnds() . . . . .	119
7.10.5.12 GetHoverStatus() [1/2] . . . . .	119
7.10.5.13 GetHoverStatus() [2/2] . . . . .	120
7.10.5.14 GetList() . . . . .	120
7.10.5.15 GetNodeDefaultPosition() . . . . .	120
7.10.5.16 GetPosition() . . . . .	120
7.10.5.17 GetShape() . . . . .	121
7.10.5.18 GetSize() . . . . .	121
7.10.5.19 GetVisible() . . . . .	121
7.10.5.20 Import() . . . . .	121
7.10.5.21 InsertNode() . . . . .	122
7.10.5.22 isSelectable() . . . . .	122
7.10.5.23 isSelected() . . . . .	123
7.10.5.24 pack() . . . . .	123
7.10.5.25 Relayout() . . . . .	123
7.10.5.26 ResetArrow() . . . . .	123
7.10.5.27 select() . . . . .	124
7.10.5.28 SetArrowType() . . . . .	124
7.10.5.29 SetCircularArrowType() . . . . .	124
7.10.5.30 SetCircularEnds() . . . . .	124
7.10.5.31 SetOrientation() . . . . .	124
7.10.5.32 SetPosition() [1/2] . . . . .	125
7.10.5.33 SetPosition() [2/2] . . . . .	125

---

7.10.5.34 SetShape()	125
7.10.5.35 SetShowHeadAndTail()	125
7.10.5.36 SetVisible()	126
7.10.5.37 size()	126
7.10.5.38 ToggleVisible()	126
7.10.5.39 UnpackAll()	126
7.10.5.40 UpdateMouseCursorWhenHover() [1/2]	126
7.10.5.41 UpdateMouseCursorWhenHover() [2/2]	127
7.10.6 Member Data Documentation	127
7.10.6.1 arrowState	127
7.10.6.2 circularArrowState	127
7.10.6.3 fonts	127
7.10.6.4 list	128
7.10.6.5 mChildren	128
7.10.6.6 mCircularEnds	128
7.10.6.7 mDisplayHeadAndTail	128
7.10.6.8 mIsSelected	128
7.10.6.9 mNodeDistance	128
7.10.6.10 mOrientation	129
7.10.6.11 mPosition	129
7.10.6.12 mShape	129
7.10.6.13 mVisible	129
7.11 State::CLLState Class Reference	130
7.11.1 Detailed Description	132
7.11.2 Member Typedef Documentation	132
7.11.2.1 Ptr	132
7.11.3 Constructor & Destructor Documentation	132
7.11.3.1 CLLState()	132
7.11.3.2 ~CLLState()	133
7.11.4 Member Function Documentation	133
7.11.4.1 AddDeleteOperation()	133
7.11.4.2 AddInitializeOperation()	133
7.11.4.3 AddInsertOperation()	133
7.11.4.4 AddIntFieldOperationOption()	134
7.11.4.5 AddNoFieldOperationOption()	134
7.11.4.6 AddOperations()	134
7.11.4.7 AddSearchOperation()	134
7.11.4.8 AddStringFieldOption()	134
7.11.4.9 AddUpdateOperation()	135
7.11.4.10 ClearAction()	135
7.11.4.11 ClearError()	135
7.11.4.12 Draw()	135

7.11.4.13 DrawCurrentActionText()	135
7.11.4.14 DrawCurrentErrorText()	135
7.11.4.15 GetContext()	136
7.11.4.16 InitNavigationBar()	136
7.11.4.17 RequestStackClear()	136
7.11.4.18 RequestStackPop()	136
7.11.4.19 RequestStackPush()	136
7.11.4.20 SetCurrentAction()	137
7.11.4.21 SetCurrentError()	137
7.11.4.22 Success()	137
7.11.4.23 Update()	137
7.11.5 Member Data Documentation	138
7.11.5.1 activeDS	138
7.11.5.2 animController	138
7.11.5.3 CLL	138
7.11.5.4 codeHighlighter	138
7.11.5.5 footer	138
7.11.5.6 mContext	139
7.11.5.7 mCurrentAction	139
7.11.5.8 mCurrentError	139
7.11.5.9 mStack	139
7.11.5.10 navigation	139
7.11.5.11 operationList	139
7.12 GUIComponent::CodeHighlighter Class Reference	140
7.12.1 Detailed Description	142
7.12.2 Member Typedef Documentation	142
7.12.2.1 Ptr	142
7.12.3 Constructor & Destructor Documentation	142
7.12.3.1 CodeHighlighter()	143
7.12.3.2 ~CodeHighlighter()	143
7.12.4 Member Function Documentation	143
7.12.4.1 AddActionDescription()	143
7.12.4.2 AddCode()	143
7.12.4.3 deselect()	143
7.12.4.4 Draw()	143
7.12.4.5 DrawActionDescription()	144
7.12.4.6 DrawCodeHighlighter()	144
7.12.4.7 GetHoverStatus() [1/2]	144
7.12.4.8 GetHoverStatus() [2/2]	144
7.12.4.9 GetPosition()	145
7.12.4.10 GetSize()	145
7.12.4.11 GetVisible()	145

---

7.12.4.12 Highlight()	146
7.12.4.13 InitButtons()	146
7.12.4.14 isSelectable()	146
7.12.4.15 isSelected()	146
7.12.4.16 select()	147
7.12.4.17 SetPosition() [1/2]	147
7.12.4.18 SetPosition() [2/2]	147
7.12.4.19 SetShowAction()	147
7.12.4.20 SetShowCode()	148
7.12.4.21 SetVisible()	148
7.12.4.22 ToggleShowAction()	148
7.12.4.23 ToggleShowCode()	148
7.12.4.24 ToggleVisible()	148
7.12.4.25 UpdateMouseCursorWhenHover() [1/2]	148
7.12.4.26 UpdateMouseCursorWhenHover() [2/2]	149
7.12.5 Member Data Documentation	149
7.12.5.1 fonts	149
7.12.5.2 mActionDescription	149
7.12.5.3 mButtonShowAction	149
7.12.5.4 mButtonShowCode	150
7.12.5.5 mCode	150
7.12.5.6 mHighlightedLine	150
7.12.5.7 mIsSelected	150
7.12.5.8 mPosition	150
7.12.5.9 mShowActionDescription	150
7.12.5.10 mShowCode	150
7.12.5.11 mVisible	151
7.13 GUI::Component Class Reference	151
7.13.1 Detailed Description	153
7.13.2 Member Typedef Documentation	153
7.13.2.1 Ptr	153
7.13.3 Constructor & Destructor Documentation	153
7.13.3.1 Component()	153
7.13.3.2 ~Component()	153
7.13.4 Member Function Documentation	154
7.13.4.1 deselect()	154
7.13.4.2 Draw()	154
7.13.4.3 GetHoverStatus() [1/2]	154
7.13.4.4 GetHoverStatus() [2/2]	155
7.13.4.5 GetPosition()	155
7.13.4.6 GetSize()	155
7.13.4.7 GetVisible()	156

---

7.13.4.8 isSelectable() . . . . .	156
7.13.4.9 isSelected() . . . . .	156
7.13.4.10 select() . . . . .	157
7.13.4.11 SetPosition() [1/2] . . . . .	157
7.13.4.12 SetPosition() [2/2] . . . . .	157
7.13.4.13 SetVisible() . . . . .	157
7.13.4.14 ToggleVisible() . . . . .	159
7.13.4.15 UpdateMouseCursorWhenHover() [1/2] . . . . .	159
7.13.4.16 UpdateMouseCursorWhenHover() [2/2] . . . . .	159
7.13.5 Member Data Documentation . . . . .	160
7.13.5.1 mIsSelected . . . . .	160
7.13.5.2 mPosition . . . . .	160
7.13.5.3 mVisible . . . . .	160
7.14 Core::List< T >::const_iterator Class Reference . . . . .	160
7.14.1 Detailed Description . . . . .	161
7.14.2 Member Typedef Documentation . . . . .	162
7.14.2.1 difference_type . . . . .	162
7.14.2.2 iterator_category . . . . .	162
7.14.2.3 pointer . . . . .	162
7.14.2.4 reference . . . . .	162
7.14.2.5 value_type . . . . .	162
7.14.3 Constructor & Destructor Documentation . . . . .	162
7.14.3.1 const_iterator() [1/4] . . . . .	163
7.14.3.2 const_iterator() [2/4] . . . . .	163
7.14.3.3 const_iterator() [3/4] . . . . .	163
7.14.3.4 const_iterator() [4/4] . . . . .	163
7.14.4 Member Function Documentation . . . . .	164
7.14.4.1 operator"!="() . . . . .	164
7.14.4.2 operator*() . . . . .	164
7.14.4.3 operator+() . . . . .	164
7.14.4.4 operator++() [1/2] . . . . .	165
7.14.4.5 operator++() [2/2] . . . . .	165
7.14.4.6 operator-() . . . . .	165
7.14.4.7 operator--() [1/2] . . . . .	166
7.14.4.8 operator--() [2/2] . . . . .	166
7.14.4.9 operator->() . . . . .	166
7.14.4.10 operator=() [1/2] . . . . .	166
7.14.4.11 operator=() [2/2] . . . . .	167
7.14.4.12 operator==() . . . . .	167
7.14.4.13 swap() . . . . .	168
7.14.5 Friends And Related Function Documentation . . . . .	168
7.14.5.1 List . . . . .	168

---

7.14.6 Member Data Documentation . . . . .	168
7.14.6.1 ptr . . . . .	168
7.15 GUI::Container Class Reference . . . . .	168
7.15.1 Detailed Description . . . . .	170
7.15.2 Member Typedef Documentation . . . . .	170
7.15.2.1 Ptr . . . . .	171
7.15.3 Constructor & Destructor Documentation . . . . .	171
7.15.3.1 Container() . . . . .	171
7.15.4 Member Function Documentation . . . . .	171
7.15.4.1 deselect() . . . . .	171
7.15.4.2 Draw() . . . . .	171
7.15.4.3 DrawCurrent() . . . . .	172
7.15.4.4 GetChildren() . . . . .	172
7.15.4.5 GetHoverStatus() [1/2] . . . . .	172
7.15.4.6 GetHoverStatus() [2/2] . . . . .	173
7.15.4.7 GetPosition() . . . . .	173
7.15.4.8 GetSize() . . . . .	173
7.15.4.9 GetVisible() . . . . .	174
7.15.4.10 isSelectable() . . . . .	174
7.15.4.11 isSelected() . . . . .	174
7.15.4.12 pack() . . . . .	174
7.15.4.13 select() . . . . .	175
7.15.4.14 SetPosition() [1/2] . . . . .	175
7.15.4.15 SetPosition() [2/2] . . . . .	175
7.15.4.16 SetVisible() . . . . .	176
7.15.4.17 ToggleVisible() . . . . .	176
7.15.4.18 UnpackAll() . . . . .	176
7.15.4.19 UpdateMouseCursorWhenHover() [1/2] . . . . .	176
7.15.4.20 UpdateMouseCursorWhenHover() [2/2] . . . . .	177
7.15.5 Member Data Documentation . . . . .	177
7.15.5.1 mChildren . . . . .	177
7.15.5.2 mIsSelected . . . . .	177
7.15.5.3 mPosition . . . . .	177
7.15.5.4 mVisible . . . . .	178
7.16 State::State::Context Struct Reference . . . . .	178
7.16.1 Detailed Description . . . . .	178
7.16.2 Constructor & Destructor Documentation . . . . .	178
7.16.2.1 Context() [1/2] . . . . .	178
7.16.2.2 Context() [2/2] . . . . .	179
7.16.3 Member Data Documentation . . . . .	179
7.16.3.1 categories . . . . .	179
7.16.3.2 dsInfo . . . . .	179

---

7.16.3.3 fonts . . . . .	179
7.16.3.4 textures . . . . .	179
7.17 Core::Deque< T > Class Template Reference . . . . .	180
7.17.1 Detailed Description . . . . .	183
7.17.2 Member Typedef Documentation . . . . .	183
7.17.2.1 List . . . . .	183
7.17.3 Member Function Documentation . . . . .	183
7.17.3.1 assign() [1/3] . . . . .	183
7.17.3.2 assign() [2/3] . . . . .	184
7.17.3.3 assign() [3/3] . . . . .	184
7.17.3.4 at() [1/2] . . . . .	184
7.17.3.5 at() [2/2] . . . . .	185
7.17.3.6 back() [1/2] . . . . .	185
7.17.3.7 back() [2/2] . . . . .	185
7.17.3.8 begin() [1/2] . . . . .	186
7.17.3.9 begin() [2/2] . . . . .	186
7.17.3.10 clear() . . . . .	186
7.17.3.11 empty() . . . . .	186
7.17.3.12 end() [1/2] . . . . .	187
7.17.3.13 end() [2/2] . . . . .	187
7.17.3.14 front() [1/2] . . . . .	187
7.17.3.15 front() [2/2] . . . . .	187
7.17.3.16 get_iterator() . . . . .	188
7.17.3.17 insert_previous() . . . . .	188
7.17.3.18 List() [1/4] . . . . .	188
7.17.3.19 List() [2/4] . . . . .	189
7.17.3.20 List() [3/4] . . . . .	189
7.17.3.21 List() [4/4] . . . . .	189
7.17.3.22 move_previous() . . . . .	189
7.17.3.23 operator=() [1/3] . . . . .	190
7.17.3.24 operator=() [2/3] . . . . .	190
7.17.3.25 operator=() [3/3] . . . . .	190
7.17.3.26 operator[]() [1/2] . . . . .	190
7.17.3.27 operator[]() [2/2] . . . . .	191
7.17.3.28 pop_back() . . . . .	191
7.17.3.29 pop_front() . . . . .	192
7.17.3.30 push_back() . . . . .	192
7.17.3.31 push_front() . . . . .	192
7.17.3.32 remove() [1/3] . . . . .	192
7.17.3.33 remove() [2/3] . . . . .	193
7.17.3.34 remove() [3/3] . . . . .	193
7.17.3.35 remove_if() [1/2] . . . . .	194

---

7.17.3.36 remove_if() [2/2]	194
7.17.3.37 reset()	194
7.17.3.38 resize() [1/2]	194
7.17.3.39 resize() [2/2]	194
7.17.3.40 reverse()	195
7.17.3.41 size()	195
7.17.3.42 swap() [1/2]	195
7.17.3.43 swap() [2/2]	196
7.17.3.44 unique() [1/2]	196
7.17.3.45 unique() [2/2]	196
7.17.4 Member Data Documentation	196
7.17.4.1 mBegin	197
7.17.4.2 mEnd	197
7.17.4.3 mSize	197
7.18 State::DLLState Class Reference	197
7.18.1 Detailed Description	200
7.18.2 Member Typedef Documentation	200
7.18.2.1 Ptr	200
7.18.3 Constructor & Destructor Documentation	200
7.18.3.1 DLLState()	200
7.18.3.2 ~DLLState()	201
7.18.4 Member Function Documentation	201
7.18.4.1 AddDeleteOperation()	201
7.18.4.2 AddInitializeOperation()	201
7.18.4.3 AddInsertOperation()	201
7.18.4.4 AddIntFieldOperationOption()	202
7.18.4.5 AddNoFieldOperationOption()	202
7.18.4.6 AddOperations()	202
7.18.4.7 AddSearchOperation()	202
7.18.4.8 AddStringFieldOption()	202
7.18.4.9 AddUpdateOperation()	203
7.18.4.10 ClearAction()	203
7.18.4.11 ClearError()	203
7.18.4.12 Draw()	203
7.18.4.13 DrawCurrentActionText()	203
7.18.4.14 DrawCurrentErrorText()	203
7.18.4.15 GetContext()	204
7.18.4.16 InitNavigationBar()	204
7.18.4.17 RequestStackClear()	204
7.18.4.18 RequestStackPop()	204
7.18.4.19 RequestStackPush()	204
7.18.4.20 SetCurrentAction()	205

7.18.4.21 SetCurrentError()	205
7.18.4.22 Success()	205
7.18.4.23 Update()	205
7.18.5 Member Data Documentation	206
7.18.5.1 activeDS	206
7.18.5.2 animController	206
7.18.5.3 codeHighlighter	206
7.18.5.4 footer	206
7.18.5.5 mContext	206
7.18.5.6 mCurrentAction	207
7.18.5.7 mCurrentError	207
7.18.5.8 mDLL	207
7.18.5.9 mStack	207
7.18.5.10 navigation	207
7.18.5.11 operationList	207
7.19 Algorithm::DoublyLinkedList Class Reference	208
7.19.1 Detailed Description	210
7.19.2 Constructor & Destructor Documentation	211
7.19.2.1 DoublyLinkedList() [1/2]	211
7.19.2.2 DoublyLinkedList() [2/2]	211
7.19.2.3 ~DoublyLinkedList()	211
7.19.3 Member Function Documentation	211
7.19.3.1 ApplyInput()	211
7.19.3.2 DeleteHead()	212
7.19.3.3 DeleteMiddle()	212
7.19.3.4 DeleteTail()	212
7.19.3.5 Empty()	212
7.19.3.6 EmptyGenerator()	213
7.19.3.7 GenerateAnimation()	213
7.19.3.8 GenerateRelayoutAnimation()	213
7.19.3.9 HighlightArrowBoth()	214
7.19.3.10 HighlightArrowNext()	214
7.19.3.11 HighlightArrowPrev()	215
7.19.3.12 InitAction()	215
7.19.3.13 InsertAfterTail()	215
7.19.3.14 InsertHead()	216
7.19.3.15 InsertMiddle()	216
7.19.3.16 Random()	216
7.19.3.17 RandomFixedSize()	216
7.19.3.18 RandomFixedSizeGenerator()	217
7.19.3.19 RandomGenerator()	217
7.19.3.20 ReadFromExternalFile()	217

---

7.19.3.21 ReadFromFileGenerator()	218
7.19.3.22 ResetVisualizer()	218
7.19.3.23 Search()	218
7.19.3.24 size()	219
7.19.3.25 Update()	219
7.19.3.26 UserDefined()	219
7.19.3.27 UserDefinedGenerator()	219
7.19.4 Member Data Documentation	220
7.19.4.1 animController	220
7.19.4.2 codeHighlighter	220
7.19.4.3 maxN	220
7.19.4.4 visualizer	221
7.20 GUIVisualizer::DoublyLinkedList Class Reference	221
7.20.1 Detailed Description	225
7.20.2 Member Typedef Documentation	225
7.20.2.1 Ptr	225
7.20.3 Member Enumeration Documentation	225
7.20.3.1 ArrowType	225
7.20.3.2 Orientation	226
7.20.4 Constructor & Destructor Documentation	226
7.20.4.1 DoublyLinkedList() [1/2]	226
7.20.4.2 DoublyLinkedList() [2/2]	226
7.20.4.3 ~DoublyLinkedList()	227
7.20.5 Member Function Documentation	227
7.20.5.1 DeleteNode()	227
7.20.5.2 deselect()	227
7.20.5.3 Draw() [1/2]	227
7.20.5.4 Draw() [2/2]	228
7.20.5.5 DrawArrow()	228
7.20.5.6 DrawCurrent()	228
7.20.5.7 GenerateNode()	229
7.20.5.8 GetArrowTypeNext()	229
7.20.5.9 GetArrowTypePrev()	229
7.20.5.10 GetChildren()	229
7.20.5.11 GetHoverStatus() [1/2]	229
7.20.5.12 GetHoverStatus() [2/2]	230
7.20.5.13 GetList()	230
7.20.5.14 GetNodeDefaultPosition()	230
7.20.5.15 GetPosition()	230
7.20.5.16 GetShape()	231
7.20.5.17 GetSize()	231
7.20.5.18 GetVisible()	231

---

7.20.5.19 Import()	231
7.20.5.20 InsertNode()	232
7.20.5.21 isSelectable()	232
7.20.5.22 isSelected()	233
7.20.5.23 pack()	233
7.20.5.24 Relayout()	233
7.20.5.25 ResetArrow()	233
7.20.5.26 select()	234
7.20.5.27 SetArrowTypeNext()	234
7.20.5.28 SetArrowTypePrev()	234
7.20.5.29 SetOrientation()	234
7.20.5.30 SetPosition() [1/2]	234
7.20.5.31 SetPosition() [2/2]	235
7.20.5.32 SetShape()	235
7.20.5.33 SetShowHeadAndTail()	235
7.20.5.34 SetVisible()	235
7.20.5.35 size()	236
7.20.5.36 ToggleVisible()	236
7.20.5.37 UnpackAll()	236
7.20.5.38 UpdateMouseCursorWhenHover() [1/2]	236
7.20.5.39 UpdateMouseCursorWhenHover() [2/2]	237
7.20.6 Member Data Documentation	237
7.20.6.1 arrowStateNext	237
7.20.6.2 arrowStatePrev	237
7.20.6.3 fonts	237
7.20.6.4 list	238
7.20.6.5 mChildren	238
7.20.6.6 mDisplayHeadAndTail	238
7.20.6.7 mIsSelected	238
7.20.6.8 mNodeDistance	238
7.20.6.9 mOrientation	238
7.20.6.10 mPosition	239
7.20.6.11 mShape	239
7.20.6.12 mVisible	239
7.21 DSInfo Class Reference	239
7.21.1 Detailed Description	240
7.21.2 Member Function Documentation	240
7.21.2.1 Get()	240
7.21.2.2 Register()	241
7.21.3 Member Data Documentation	241
7.21.3.1 mFactories	241
7.22 Algorithm::DynamicArray Class Reference	242

---

7.22.1 Detailed Description . . . . .	244
7.22.2 Constructor & Destructor Documentation . . . . .	244
7.22.2.1 DynamicArray() [1/2] . . . . .	244
7.22.2.2 DynamicArray() [2/2] . . . . .	244
7.22.2.3 ~DynamicArray() . . . . .	245
7.22.3 Member Function Documentation . . . . .	245
7.22.3.1 Access() . . . . .	245
7.22.3.2 ApplyInput() . . . . .	245
7.22.3.3 Empty() . . . . .	246
7.22.3.4 EmptyGenerator() . . . . .	246
7.22.3.5 GenerateAnimation() . . . . .	246
7.22.3.6 GenerateRelayoutAnimation() . . . . .	247
7.22.3.7 InitAction() . . . . .	247
7.22.3.8 PopBack() . . . . .	247
7.22.3.9 Push() . . . . .	247
7.22.3.10 PushBack() . . . . .	248
7.22.3.11 Random() . . . . .	248
7.22.3.12 RandomFixedSize() . . . . .	248
7.22.3.13 RandomFixedSizeGenerator() . . . . .	249
7.22.3.14 RandomGenerator() . . . . .	249
7.22.3.15 ReadFromExternalFile() . . . . .	249
7.22.3.16 ReadFromFileGenerator() . . . . .	250
7.22.3.17 Remove() . . . . .	250
7.22.3.18 ResetVisualizer() . . . . .	250
7.22.3.19 Search() . . . . .	251
7.22.3.20 size() . . . . .	251
7.22.3.21 Update() . . . . .	251
7.22.3.22 UserDefined() . . . . .	251
7.22.3.23 UserDefinedGenerator() . . . . .	252
7.22.4 Member Data Documentation . . . . .	252
7.22.4.1 animController . . . . .	252
7.22.4.2 codeHighlighter . . . . .	252
7.22.4.3 maxN . . . . .	253
7.22.4.4 visualizer . . . . .	253
7.23 GUIVisualizer::DynamicArray Class Reference . . . . .	253
7.23.1 Detailed Description . . . . .	257
7.23.2 Member Typedef Documentation . . . . .	257
7.23.2.1 Ptr . . . . .	257
7.23.3 Constructor & Destructor Documentation . . . . .	257
7.23.3.1 DynamicArray() [1/2] . . . . .	257
7.23.3.2 DynamicArray() [2/2] . . . . .	257
7.23.3.3 ~DynamicArray() . . . . .	258

7.23.4 Member Function Documentation . . . . .	258
7.23.4.1 Clear() . . . . .	258
7.23.4.2 DeleteNode() . . . . .	258
7.23.4.3 deselect() . . . . .	258
7.23.4.4 Draw() [1/2] . . . . .	259
7.23.4.5 Draw() [2/2] . . . . .	259
7.23.4.6 DrawCurrent() . . . . .	259
7.23.4.7 GenerateNode() . . . . .	260
7.23.4.8 GetCapacity() . . . . .	260
7.23.4.9 GetCapacityFromLength() . . . . .	260
7.23.4.10 GetChildren() . . . . .	260
7.23.4.11 GetHoverStatus() [1/2] . . . . .	260
7.23.4.12 GetHoverStatus() [2/2] . . . . .	261
7.23.4.13 GetLength() . . . . .	261
7.23.4.14 GetList() . . . . .	261
7.23.4.15 GetNodeDefaultPosition() . . . . .	262
7.23.4.16 GetPosition() . . . . .	262
7.23.4.17 GetShape() . . . . .	262
7.23.4.18 GetSize() . . . . .	262
7.23.4.19 GetVisible() . . . . .	263
7.23.4.20 Import() . . . . .	263
7.23.4.21 InsertNode() . . . . .	263
7.23.4.22 isSelectable() . . . . .	264
7.23.4.23 isSelected() . . . . .	264
7.23.4.24 operator[]() [1/2] . . . . .	264
7.23.4.25 operator[]() [2/2] . . . . .	265
7.23.4.26 pack() . . . . .	265
7.23.4.27 Relayout() . . . . .	265
7.23.4.28 Reserve() . . . . .	266
7.23.4.29 Resize() . . . . .	267
7.23.4.30 select() . . . . .	267
7.23.4.31 SetPosition() [1/2] . . . . .	267
7.23.4.32 SetPosition() [2/2] . . . . .	268
7.23.4.33 SetShape() . . . . .	268
7.23.4.34 SetVisible() . . . . .	268
7.23.4.35 ToggleVisible() . . . . .	268
7.23.4.36 UnpackAll() . . . . .	269
7.23.4.37 UpdateMouseCursorWhenHover() [1/2] . . . . .	269
7.23.4.38 UpdateMouseCursorWhenHover() [2/2] . . . . .	269
7.23.5 Member Data Documentation . . . . .	270
7.23.5.1 capacity . . . . .	270
7.23.5.2 fonts . . . . .	270

---

7.23.5.3 length . . . . .	270
7.23.5.4 list . . . . .	270
7.23.5.5 mChildren . . . . .	270
7.23.5.6 mIsSelected . . . . .	270
7.23.5.7 mNodeDistance . . . . .	271
7.23.5.8 mPosition . . . . .	271
7.23.5.9 mShape . . . . .	271
7.23.5.10 mVisible . . . . .	271
7.24 State::DynamicArrayState Class Reference . . . . .	271
7.24.1 Detailed Description . . . . .	274
7.24.2 Member Typedef Documentation . . . . .	274
7.24.2.1 Ptr . . . . .	274
7.24.3 Constructor & Destructor Documentation . . . . .	274
7.24.3.1 DynamicArrayState() . . . . .	274
7.24.3.2 ~DynamicArrayState() . . . . .	275
7.24.4 Member Function Documentation . . . . .	275
7.24.4.1 AddAccessOperation() . . . . .	275
7.24.4.2 AddDeleteOperation() . . . . .	275
7.24.4.3 AddInitializeOperation() . . . . .	275
7.24.4.4 AddInsertOperation() . . . . .	276
7.24.4.5 AddIntFieldOperationOption() . . . . .	276
7.24.4.6 AddNoFieldOperationOption() . . . . .	276
7.24.4.7 AddOperations() . . . . .	276
7.24.4.8 AddSearchOperation() . . . . .	276
7.24.4.9 AddStringFieldOption() . . . . .	277
7.24.4.10 AddUpdateOperation() . . . . .	277
7.24.4.11 ClearAction() . . . . .	277
7.24.4.12 ClearError() . . . . .	277
7.24.4.13 Draw() . . . . .	277
7.24.4.14 DrawCurrentActionText() . . . . .	278
7.24.4.15 DrawCurrentErrorText() . . . . .	278
7.24.4.16 GetContext() . . . . .	278
7.24.4.17 InitNavigationBar() . . . . .	278
7.24.4.18 RequestStackClear() . . . . .	278
7.24.4.19 RequestStackPop() . . . . .	279
7.24.4.20 RequestStackPush() . . . . .	279
7.24.4.21 SetCurrentAction() . . . . .	279
7.24.4.22 SetCurrentError() . . . . .	279
7.24.4.23 Success() . . . . .	280
7.24.4.24 Update() . . . . .	280
7.24.5 Member Data Documentation . . . . .	280
7.24.5.1 activeDS . . . . .	280

---

7.24.5.2 animController . . . . .	280
7.24.5.3 codeHighlighter . . . . .	281
7.24.5.4 footer . . . . .	281
7.24.5.5 mContext . . . . .	281
7.24.5.6 mCurrentAction . . . . .	281
7.24.5.7 mCurrentError . . . . .	281
7.24.5.8 mDynamicArray . . . . .	281
7.24.5.9 mStack . . . . .	281
7.24.5.10 navigation . . . . .	282
7.24.5.11 operationList . . . . .	282
7.25 Fontholder Class Reference . . . . .	282
7.25.1 Detailed Description . . . . .	283
7.25.2 Constructor & Destructor Documentation . . . . .	283
7.25.2.1 FontHolder() . . . . .	283
7.25.2.2 ~FontHolder() . . . . .	284
7.25.3 Member Function Documentation . . . . .	284
7.25.3.1 Get() [1/2] . . . . .	284
7.25.3.2 Get() [2/2] . . . . .	284
7.25.3.3 InsertResource() . . . . .	285
7.25.3.4 Load() . . . . .	285
7.25.4 Member Data Documentation . . . . .	286
7.25.4.1 mFontMap . . . . .	286
7.26 GUIComponent::Footer< T > Class Template Reference . . . . .	286
7.26.1 Detailed Description . . . . .	289
7.26.2 Member Typedef Documentation . . . . .	289
7.26.2.1 Ptr . . . . .	289
7.26.3 Constructor & Destructor Documentation . . . . .	289
7.26.3.1 Footer() . . . . .	289
7.26.3.2 ~Footer() . . . . .	290
7.26.4 Member Function Documentation . . . . .	290
7.26.4.1 deselect() . . . . .	290
7.26.4.2 Draw() [1/2] . . . . .	290
7.26.4.3 Draw() [2/2] . . . . .	290
7.26.4.4 DrawCurrent() . . . . .	291
7.26.4.5 GetChildren() . . . . .	291
7.26.4.6 GetHoverStatus() [1/2] . . . . .	291
7.26.4.7 GetHoverStatus() [2/2] . . . . .	291
7.26.4.8 GetPosition() . . . . .	292
7.26.4.9 GetSize() . . . . .	292
7.26.4.10 GetVisible() . . . . .	292
7.26.4.11 IsSelectable() . . . . .	293
7.26.4.12 IsSelected() . . . . .	293

---

7.26.4.13 pack()	293
7.26.4.14 select()	294
7.26.4.15 SetPosition() [1/2]	294
7.26.4.16 SetPosition() [2/2]	294
7.26.4.17 SetVisible()	295
7.26.4.18 ToggleVisible()	295
7.26.4.19 UnpackAll()	295
7.26.4.20 UpdateMouseCursorWhenHover() [1/2]	295
7.26.4.21 UpdateMouseCursorWhenHover() [2/2]	296
7.26.5 Member Data Documentation	296
7.26.5.1 mChildren	296
7.26.5.2 mIsSelected	296
7.26.5.3 mPosition	296
7.26.5.4 mVisible	297
7.27 State::HomepageState Class Reference	297
7.27.1 Detailed Description	299
7.27.2 Member Typedef Documentation	300
7.27.2.1 Ptr	300
7.27.3 Constructor & Destructor Documentation	300
7.27.3.1 HomepageState()	300
7.27.3.2 ~HomepageState()	300
7.27.4 Member Function Documentation	300
7.27.4.1 CreateCard()	300
7.27.4.2 Draw()	301
7.27.4.3 DrawIntroduction()	301
7.27.4.4 GetContext()	301
7.27.4.5 InitCards()	301
7.27.4.6 InitNavigationBar()	301
7.27.4.7 RequestStackClear()	302
7.27.4.8 RequestStackPop()	302
7.27.4.9 RequestStackPush()	302
7.27.4.10 Update()	302
7.27.5 Member Data Documentation	303
7.27.5.1 hasInitializeCard	303
7.27.5.2 mCards	303
7.27.5.3 mContext	303
7.27.5.4 mStack	303
7.27.5.5 navigation	304
7.28 CategoryInfo::Info Struct Reference	304
7.28.1 Detailed Description	304
7.28.2 Constructor & Destructor Documentation	304
7.28.2.1 Info()	304

---

7.28.3 Member Data Documentation . . . . .	305
7.28.3.1 categoryID . . . . .	305
7.28.3.2 categoryName . . . . .	305
7.28.3.3 mDS . . . . .	305
7.29 DSInfo::Info Struct Reference . . . . .	306
7.29.1 Detailed Description . . . . .	306
7.29.2 Constructor & Destructor Documentation . . . . .	306
7.29.2.1 Info() . . . . .	306
7.29.3 Member Data Documentation . . . . .	307
7.29.3.1 abbr . . . . .	307
7.29.3.2 categoryID . . . . .	307
7.29.3.3 ID . . . . .	307
7.29.3.4 name . . . . .	308
7.29.3.5 stateID . . . . .	308
7.29.3.6 thumbnail . . . . .	308
7.30 GUIComponent::InputField Class Reference . . . . .	309
7.30.1 Detailed Description . . . . .	311
7.30.2 Member Typedef Documentation . . . . .	311
7.30.2.1 Ptr . . . . .	311
7.30.3 Constructor & Destructor Documentation . . . . .	311
7.30.3.1 InputField() . . . . .	311
7.30.3.2 ~InputField() . . . . .	312
7.30.4 Member Function Documentation . . . . .	312
7.30.4.1 AllFieldDisableEdit() . . . . .	312
7.30.4.2 deselect() . . . . .	312
7.30.4.3 Draw() . . . . .	312
7.30.4.4 DrawField() . . . . .	312
7.30.4.5 ExtractValue() . . . . .	313
7.30.4.6 GetEditMode() . . . . .	313
7.30.4.7 GetHoverStatus() [1/2] . . . . .	313
7.30.4.8 GetHoverStatus() [2/2] . . . . .	313
7.30.4.9 GetLabel() . . . . .	314
7.30.4.10 GetPosition() . . . . .	314
7.30.4.11 GetSize() . . . . .	314
7.30.4.12 GetVisible() . . . . .	314
7.30.4.13 IsClicked() . . . . .	315
7.30.4.14 IsSelectable() . . . . .	315
7.30.4.15 IsSelected() . . . . .	315
7.30.4.16 Randomize() . . . . .	315
7.30.4.17 select() . . . . .	316
7.30.4.18 SetEditMode() . . . . .	316
7.30.4.19 SetInputFieldSize() . . . . .	316

---

7.30.4.20 SetLabel()	316
7.30.4.21 SetLabelSize()	316
7.30.4.22 SetPosition() [1/2]	316
7.30.4.23 SetPosition() [2/2]	317
7.30.4.24 SetVisible()	317
7.30.4.25 ToggleVisible()	317
7.30.4.26 UpdateMouseCursorWhenHover() [1/2]	318
7.30.4.27 UpdateMouseCursorWhenHover() [2/2]	318
7.30.5 Member Data Documentation	318
7.30.5.1 editMode	318
7.30.5.2 extractedValue	318
7.30.5.3 fields	319
7.30.5.4 fonts	319
7.30.5.5 inputFieldSize	319
7.30.5.6 label	319
7.30.5.7 labelFontSize	319
7.30.5.8 mFieldIndex	319
7.30.5.9 mIsSelected	319
7.30.5.10 mPosition	320
7.30.5.11 mVisible	320
7.31 State::ArrayState< T >::IntegerInput Struct Reference	320
7.31.1 Member Data Documentation	320
7.31.1.1 label	320
7.31.1.2 maxValue	321
7.31.1.3 minValue	321
7.31.1.4 width	321
7.32 State::LLState< T >::IntegerInput Struct Reference	321
7.32.1 Member Data Documentation	321
7.32.1.1 label	321
7.32.1.2 maxValue	322
7.32.1.3 minValue	322
7.32.1.4 width	322
7.33 GUIComponent::IntegerField Class Reference	322
7.33.1 Detailed Description	325
7.33.2 Member Typedef Documentation	325
7.33.2.1 Ptr	325
7.33.3 Constructor & Destructor Documentation	325
7.33.3.1 IntegerInputField()	325
7.33.3.2 ~IntegerInputField()	326
7.33.4 Member Function Documentation	326
7.33.4.1 AllFieldDisableEdit()	326
7.33.4.2 deselect()	326

---

7.33.4.3 Draw()	326
7.33.4.4 DrawField()	326
7.33.4.5 ExtractValue()	327
7.33.4.6 GetEditMode()	327
7.33.4.7 GetHoverStatus() [1/2]	327
7.33.4.8 GetHoverStatus() [2/2]	327
7.33.4.9 GetLabel()	328
7.33.4.10GetPosition()	328
7.33.4.11 GetSize()	328
7.33.4.12 GetVisible()	328
7.33.4.13 IsClicked()	329
7.33.4.14 isSelectable()	329
7.33.4.15 isSelected()	329
7.33.4.16 Randomize()	329
7.33.4.17 select()	330
7.33.4.18 SetConstraint()	330
7.33.4.19 SetEditMode()	330
7.33.4.20 SetInputFieldSize()	330
7.33.4.21 SetLabel()	330
7.33.4.22 SetLabelSize()	330
7.33.4.23 SetPosition() [1/2]	330
7.33.4.24 SetPosition() [2/2]	331
7.33.4.25 SetVisible()	331
7.33.4.26 ToggleVisible()	331
7.33.4.27 UpdateMouseCursorWhenHover() [1/2]	332
7.33.4.28 UpdateMouseCursorWhenHover() [2/2]	332
7.33.5 Member Data Documentation	332
7.33.5.1 editMode	332
7.33.5.2 extractedValue	332
7.33.5.3 fields	333
7.33.5.4 fonts	333
7.33.5.5 input	333
7.33.5.6 inputFieldSize	333
7.33.5.7 label	333
7.33.5.8 labelFontSize	333
7.33.5.9 mFieldIndex	333
7.33.5.10 mIsSelected	334
7.33.5.11 mMaxValue	334
7.33.5.12 mMinValue	334
7.33.5.13 mPosition	334
7.33.5.14 mVisible	334
7.34 Core::List< T >::iterator Class Reference	334

---

---

7.34.1 Detailed Description . . . . .	335
7.34.2 Member Typedef Documentation . . . . .	336
7.34.2.1 difference_type . . . . .	336
7.34.2.2 iterator_category . . . . .	336
7.34.2.3 pointer . . . . .	336
7.34.2.4 reference . . . . .	336
7.34.2.5 value_type . . . . .	336
7.34.3 Constructor & Destructor Documentation . . . . .	336
7.34.3.1 iterator() [1/2] . . . . .	337
7.34.3.2 iterator() [2/2] . . . . .	337
7.34.4 Member Function Documentation . . . . .	337
7.34.4.1 operator"!=()" . . . . .	337
7.34.4.2 operator*() . . . . .	337
7.34.4.3 operator+() . . . . .	338
7.34.4.4 operator++() [1/2] . . . . .	338
7.34.4.5 operator++() [2/2] . . . . .	338
7.34.4.6 operator-() . . . . .	338
7.34.4.7 operator--() [1/2] . . . . .	339
7.34.4.8 operator--() [2/2] . . . . .	339
7.34.4.9 operator->() . . . . .	339
7.34.4.10 operator=(()) . . . . .	339
7.34.4.11 operator==(()) . . . . .	340
7.34.4.12 swap() . . . . .	340
7.34.5 Friends And Related Function Documentation . . . . .	340
7.34.5.1 List . . . . .	341
7.34.6 Member Data Documentation . . . . .	341
7.34.6.1 ptr . . . . .	341
7.35 GUIVisualizer::LinkedList Class Reference . . . . .	341
7.35.1 Detailed Description . . . . .	345
7.35.2 Member Typedef Documentation . . . . .	345
7.35.2.1 Ptr . . . . .	345
7.35.3 Member Enumeration Documentation . . . . .	345
7.35.3.1 ArrowType . . . . .	345
7.35.3.2 Orientation . . . . .	346
7.35.4 Constructor & Destructor Documentation . . . . .	346
7.35.4.1 LinkedList() [1/2] . . . . .	346
7.35.4.2 LinkedList() [2/2] . . . . .	346
7.35.4.3 ~LinkedList() . . . . .	346
7.35.5 Member Function Documentation . . . . .	347
7.35.5.1 DeleteNode() . . . . .	347
7.35.5.2 deselect() . . . . .	347
7.35.5.3 Draw() [1/2] . . . . .	347

7.35.5.4 Draw() [2/2] . . . . .	348
7.35.5.5 DrawCurrent() . . . . .	348
7.35.5.6 GenerateNode() . . . . .	348
7.35.5.7 GetChildren() . . . . .	348
7.35.5.8 GetHoverStatus() [1/2] . . . . .	349
7.35.5.9 GetHoverStatus() [2/2] . . . . .	349
7.35.5.10 GetList() . . . . .	349
7.35.5.11 GetNodeDefaultPosition() . . . . .	349
7.35.5.12 GetPosition() . . . . .	350
7.35.5.13 GetShape() . . . . .	350
7.35.5.14 GetSize() . . . . .	350
7.35.5.15 GetVisible() . . . . .	350
7.35.5.16 Import() . . . . .	350
7.35.5.17 InsertNode() . . . . .	351
7.35.5.18 isSelectable() . . . . .	351
7.35.5.19 isSelected() . . . . .	352
7.35.5.20 pack() . . . . .	352
7.35.5.21 Relayout() . . . . .	352
7.35.5.22 select() . . . . .	352
7.35.5.23 SetOrientation() . . . . .	353
7.35.5.24 SetPosition() [1/2] . . . . .	353
7.35.5.25 SetPosition() [2/2] . . . . .	353
7.35.5.26 SetShape() . . . . .	353
7.35.5.27 SetShowHeadAndTail() . . . . .	354
7.35.5.28 SetVisible() . . . . .	354
7.35.5.29 size() . . . . .	354
7.35.5.30 ToggleVisible() . . . . .	354
7.35.5.31 UnpackAll() . . . . .	355
7.35.5.32 UpdateMouseCursorWhenHover() [1/2] . . . . .	355
7.35.5.33 UpdateMouseCursorWhenHover() [2/2] . . . . .	355
7.35.6 Member Data Documentation . . . . .	355
7.35.6.1 fonts . . . . .	356
7.35.6.2 list . . . . .	356
7.35.6.3 mChildren . . . . .	356
7.35.6.4 mDisplayHeadAndTail . . . . .	356
7.35.6.5 mIsSelected . . . . .	356
7.35.6.6 mNodeDistance . . . . .	356
7.35.6.7 mOrientation . . . . .	357
7.35.6.8 mPosition . . . . .	357
7.35.6.9 mShape . . . . .	357
7.35.6.10 mVisible . . . . .	357
7.36 Core::List< T > Class Template Reference . . . . .	358

7.36.1 Detailed Description . . . . .	360
7.36.2 Constructor & Destructor Documentation . . . . .	361
7.36.2.1 List() [1/4] . . . . .	361
7.36.2.2 List() [2/4] . . . . .	361
7.36.2.3 ~List() . . . . .	361
7.36.2.4 List() [3/4] . . . . .	361
7.36.2.5 List() [4/4] . . . . .	362
7.36.3 Member Function Documentation . . . . .	362
7.36.3.1 assign() [1/3] . . . . .	362
7.36.3.2 assign() [2/3] . . . . .	362
7.36.3.3 assign() [3/3] . . . . .	363
7.36.3.4 at() [1/2] . . . . .	363
7.36.3.5 at() [2/2] . . . . .	363
7.36.3.6 back() [1/2] . . . . .	364
7.36.3.7 back() [2/2] . . . . .	364
7.36.3.8 begin() [1/2] . . . . .	365
7.36.3.9 begin() [2/2] . . . . .	365
7.36.3.10 clear() . . . . .	365
7.36.3.11 empty() . . . . .	365
7.36.3.12 end() [1/2] . . . . .	365
7.36.3.13 end() [2/2] . . . . .	366
7.36.3.14 front() [1/2] . . . . .	366
7.36.3.15 front() [2/2] . . . . .	366
7.36.3.16 get_iterator() . . . . .	366
7.36.3.17 insert_previous() . . . . .	367
7.36.3.18 move_previous() . . . . .	367
7.36.3.19 operator=() [1/3] . . . . .	367
7.36.3.20 operator=() [2/3] . . . . .	368
7.36.3.21 operator=() [3/3] . . . . .	368
7.36.3.22 operator[]() [1/2] . . . . .	368
7.36.3.23 operator[]() [2/2] . . . . .	369
7.36.3.24 pop_back() . . . . .	369
7.36.3.25 pop_front() . . . . .	369
7.36.3.26 push_back() . . . . .	370
7.36.3.27 push_front() . . . . .	370
7.36.3.28 remove() [1/3] . . . . .	370
7.36.3.29 remove() [2/3] . . . . .	371
7.36.3.30 remove() [3/3] . . . . .	371
7.36.3.31 remove_if() [1/2] . . . . .	371
7.36.3.32 remove_if() [2/2] . . . . .	372
7.36.3.33 reset() . . . . .	372
7.36.3.34 resize() [1/2] . . . . .	372

---

7.36.3.35 resize() [2/2]	372
7.36.3.36 reverse()	373
7.36.3.37 size()	373
7.36.3.38 swap()	373
7.36.3.39 unique() [1/2]	373
7.36.3.40 unique() [2/2]	374
7.36.4 Member Data Documentation	374
7.36.4.1 mBegin	374
7.36.4.2 mEnd	374
7.36.4.3 mSize	374
7.37 State::LLState< T > Class Template Reference	375
7.37.1 Detailed Description	377
7.37.2 Member Typedef Documentation	377
7.37.2.1 Ptr	377
7.37.3 Constructor & Destructor Documentation	377
7.37.3.1 LLState()	378
7.37.3.2 ~LLState()	379
7.37.4 Member Function Documentation	379
7.37.4.1 AddDeleteOperation()	379
7.37.4.2 AddInitializeOperation()	379
7.37.4.3 AddInsertOperation()	379
7.37.4.4 AddIntFieldOperationOption()	380
7.37.4.5 AddNoFieldOperationOption()	380
7.37.4.6 AddOperations()	380
7.37.4.7 AddSearchOperation()	380
7.37.4.8 AddStringFieldOption()	381
7.37.4.9 AddUpdateOperation()	381
7.37.4.10 ClearAction()	381
7.37.4.11 ClearError()	381
7.37.4.12 Draw()	381
7.37.4.13 DrawCurrentActionText()	382
7.37.4.14 DrawCurrentErrorText()	382
7.37.4.15 GetContext()	382
7.37.4.16 InitNavigationBar()	382
7.37.4.17 RequestStackClear()	382
7.37.4.18 RequestStackPop()	383
7.37.4.19 RequestStackPush()	383
7.37.4.20 SetCurrentAction()	383
7.37.4.21 SetCurrentError()	383
7.37.4.22 Success()	385
7.37.4.23 Update()	385
7.37.5 Member Data Documentation	385

---

7.37.5.1 activeDS . . . . .	385
7.37.5.2 animController . . . . .	386
7.37.5.3 codeHighlighter . . . . .	386
7.37.5.4 footer . . . . .	386
7.37.5.5 mContext . . . . .	386
7.37.5.6 mCurrentAction . . . . .	386
7.37.5.7 mCurrentError . . . . .	386
7.37.5.8 mStack . . . . .	386
7.37.5.9 navigation . . . . .	387
7.37.5.10 operationList . . . . .	387
7.38 GUIComponent::NavigationBar Class Reference . . . . .	387
7.38.1 Detailed Description . . . . .	390
7.38.2 Member Typedef Documentation . . . . .	390
7.38.2.1 Ptr . . . . .	390
7.38.3 Constructor & Destructor Documentation . . . . .	391
7.38.3.1 NavigationBar() [1/2] . . . . .	391
7.38.3.2 NavigationBar() [2/2] . . . . .	391
7.38.3.3 ~NavigationBar() . . . . .	391
7.38.4 Member Function Documentation . . . . .	391
7.38.4.1 ClearTitle() . . . . .	391
7.38.4.2 deselect() . . . . .	391
7.38.4.3 Draw() . . . . .	391
7.38.4.4 DrawLogo() . . . . .	392
7.38.4.5 DrawSwitchTheme() . . . . .	392
7.38.4.6 DrawTitles() . . . . .	392
7.38.4.7 GetHoverStatus() [1/2] . . . . .	392
7.38.4.8 GetHoverStatus() [2/2] . . . . .	392
7.38.4.9GetPosition() . . . . .	393
7.38.4.10GetSize() . . . . .	393
7.38.4.11GetVisible() . . . . .	393
7.38.4.12InsertTitle() . . . . .	394
7.38.4.13isSelectable() . . . . .	394
7.38.4.14isSelected() . . . . .	394
7.38.4.15select() . . . . .	395
7.38.4.16SetActiveTitle() . . . . .	395
7.38.4.17SetCategory() . . . . .	395
7.38.4.18SetDirectLink() . . . . .	395
7.38.4.19SetHomepageID() . . . . .	395
7.38.4.20SetPosition() [1/2] . . . . .	395
7.38.4.21SetPosition() [2/2] . . . . .	396
7.38.4.22SetVisibleTitle() . . . . .	396
7.38.4.23SetVisible() . . . . .	396

---

7.38.4.24 ToggleVisible() . . . . .	397
7.38.4.25 UpdateMouseCursorWhenHover() [1/2] . . . . .	397
7.38.4.26 UpdateMouseCursorWhenHover() [2/2] . . . . .	397
7.38.5 Member Data Documentation . . . . .	397
7.38.5.1 activeTitle . . . . .	397
7.38.5.2 currentCategory . . . . .	398
7.38.5.3 fonts . . . . .	398
7.38.5.4 hasTitle . . . . .	398
7.38.5.5 homepageIndex . . . . .	398
7.38.5.6 hoverBounds . . . . .	398
7.38.5.7 isHover . . . . .	398
7.38.5.8 mIsSelected . . . . .	398
7.38.5.9 mPosition . . . . .	399
7.38.5.10 mTitles . . . . .	399
7.38.5.11 mVisible . . . . .	399
7.38.5.12 toLink . . . . .	399
7.39 Core::Node< T > Class Template Reference . . . . .	399
7.39.1 Detailed Description . . . . .	400
7.39.2 Constructor & Destructor Documentation . . . . .	400
7.39.2.1 Node() [1/4] . . . . .	400
7.39.2.2 Node() [2/4] . . . . .	400
7.39.2.3 Node() [3/4] . . . . .	401
7.39.2.4 Node() [4/4] . . . . .	401
7.39.3 Member Data Documentation . . . . .	401
7.39.3.1 mNext . . . . .	401
7.39.3.2 mPrev . . . . .	402
7.39.3.3 mValue . . . . .	402
7.40 GUIVisualizer::Node Class Reference . . . . .	402
7.40.1 Detailed Description . . . . .	406
7.40.2 Member Typedef Documentation . . . . .	406
7.40.2.1 Ptr . . . . .	406
7.40.3 Member Enumeration Documentation . . . . .	406
7.40.3.1 Shape . . . . .	406
7.40.3.2 State . . . . .	407
7.40.4 Constructor & Destructor Documentation . . . . .	407
7.40.4.1 Node() [1/2] . . . . .	407
7.40.4.2 Node() [2/2] . . . . .	408
7.40.4.3 ~Node() . . . . .	408
7.40.5 Member Function Documentation . . . . .	408
7.40.5.1 AddColor() . . . . .	408
7.40.5.2 AnimationOnNode() . . . . .	408
7.40.5.3 ClearLabel() . . . . .	408

---

7.40.5.4 deselect()	409
7.40.5.5 Draw() [1/2]	409
7.40.5.6 Draw() [2/2]	409
7.40.5.7 DrawLabel()	409
7.40.5.8 DrawNode()	410
7.40.5.9 GetBackgroundColor()	410
7.40.5.10 GetHoverStatus() [1/2]	410
7.40.5.11 GetHoverStatus() [2/2]	410
7.40.5.12 GetNodeState()	411
7.40.5.13 GetOutlineColor()	411
7.40.5.14 GetPosition()	411
7.40.5.15 GetReachable()	411
7.40.5.16 GetShape()	411
7.40.5.17 GetSize()	412
7.40.5.18 GetTextColor()	412
7.40.5.19 GetValue()	412
7.40.5.20 GetVisible()	412
7.40.5.21 IsActive()	413
7.40.5.22 isSelectable()	413
7.40.5.23 isSelected()	413
7.40.5.24 select()	413
7.40.5.25 SetActive()	413
7.40.5.26 SetLabel()	414
7.40.5.27 SetLabelFontSize()	414
7.40.5.28 SetNodeState()	414
7.40.5.29 SetPosition() [1/2]	414
7.40.5.30 SetPosition() [2/2]	415
7.40.5.31 SetRadius()	415
7.40.5.32 SetReachable()	415
7.40.5.33 SetShape()	415
7.40.5.34 SetValue()	415
7.40.5.35 SetValueFontSize()	416
7.40.5.36 SetVisible()	416
7.40.5.37 ToggleVisible()	416
7.40.5.38 UpdateMouseCursorWhenHover() [1/2]	416
7.40.5.39 UpdateMouseCursorWhenHover() [2/2]	417
40.6 Member Data Documentation	417
7.40.6.1 animateNode	417
7.40.6.2 fonts	417
7.40.6.3 labelFontSize	417
7.40.6.4 mActive	418
7.40.6.5 mActiveColor	418

---

7.40.6.6 mBackgroundColor . . . . .	418
7.40.6.7 mBorderColor . . . . .	418
7.40.6.8 mDefaultColor . . . . .	418
7.40.6.9 mIsSelected . . . . .	418
7.40.6.10 mLabel . . . . .	418
7.40.6.11 mNodeState . . . . .	419
7.40.6.12 mOutlineColor . . . . .	419
7.40.6.13 mPosition . . . . .	419
7.40.6.14 mRadius . . . . .	419
7.40.6.15 mReachable . . . . .	419
7.40.6.16 mShape . . . . .	419
7.40.6.17 mTextColor . . . . .	419
7.40.6.18 mValue . . . . .	420
7.40.6.19 mVisible . . . . .	420
7.40.6.20 valueFontSize . . . . .	420
7.41 NonCopyable< T > Class Template Reference . . . . .	420
7.41.1 Detailed Description . . . . .	420
7.41.2 Constructor & Destructor Documentation . . . . .	421
7.41.2.1 NonCopyable() [1/2] . . . . .	421
7.41.2.2 NonCopyable() [2/2] . . . . .	421
7.41.2.3 ~NonCopyable() . . . . .	421
7.41.3 Member Function Documentation . . . . .	421
7.41.3.1 operator=() . . . . .	422
7.42 GUIComponent::OperationContainer Class Reference . . . . .	422
7.42.1 Detailed Description . . . . .	425
7.42.2 Member Typedef Documentation . . . . .	425
7.42.2.1 Ptr . . . . .	425
7.42.3 Constructor & Destructor Documentation . . . . .	425
7.42.3.1 OperationContainer() . . . . .	425
7.42.3.2 ~OperationContainer() . . . . .	425
7.42.4 Member Function Documentation . . . . .	425
7.42.4.1 deselect() . . . . .	426
7.42.4.2 Draw() . . . . .	426
7.42.4.3 DrawCurrent() . . . . .	426
7.42.4.4 GetChildren() . . . . .	426
7.42.4.5 GetHoverStatus() [1/2] . . . . .	427
7.42.4.6 GetHoverStatus() [2/2] . . . . .	427
7.42.4.7 GetPosition() . . . . .	427
7.42.4.8 GetSize() . . . . .	428
7.42.4.9 GetVisible() . . . . .	428
7.42.4.10 IsSelectable() . . . . .	428
7.42.4.11 isSelected() . . . . .	429

---

7.42.4.12 pack()	429
7.42.4.13 select()	429
7.42.4.14 SetPosition() [1/2]	429
7.42.4.15 SetPosition() [2/2]	430
7.42.4.16 SetVisible()	430
7.42.4.17 ToggleVisible()	430
7.42.4.18 UnpackAll()	431
7.42.4.19 UpdateMouseCursorWhenHover() [1/2]	431
7.42.4.20 UpdateMouseCursorWhenHover() [2/2]	431
7.42.4.21 UpdatePosition()	432
7.42.5 Member Data Documentation	432
7.42.5.1 mChildren	432
7.42.5.2 mIsSelected	432
7.42.5.3 mPosition	432
7.42.5.4 mVisible	432
7.43 GUIComponent::OperationList Class Reference	433
7.43.1 Detailed Description	435
7.43.2 Member Typedef Documentation	435
7.43.2.1 Ptr	435
7.43.3 Constructor & Destructor Documentation	436
7.43.3.1 OperationList() [1/2]	436
7.43.3.2 OperationList() [2/2]	436
7.43.3.3 ~OperationList()	436
7.43.4 Member Function Documentation	436
7.43.4.1 AddOperation()	436
7.43.4.2 deselect()	436
7.43.4.3 Draw()	436
7.43.4.4 DrawCurrent()	437
7.43.4.5 GetChildren()	437
7.43.4.6 GetHoverStatus() [1/2]	437
7.43.4.7 GetHoverStatus() [2/2]	438
7.43.4.8 GetPosition()	438
7.43.4.9 GetSize()	438
7.43.4.10 GetVisible()	439
7.43.4.11 HideAllOptions()	439
7.43.4.12 InitActionBar()	439
7.43.4.13 isSelectable()	439
7.43.4.14 isSelected()	440
7.43.4.15 pack()	440
7.43.4.16 select()	440
7.43.4.17 SetPosition() [1/2]	440
7.43.4.18 SetPosition() [2/2]	441

7.43.4.19 SetVisible()	441
7.43.4.20 ShowOptions()	441
7.43.4.21 ToggleOperations()	442
7.43.4.22 ToggleVisible()	442
7.43.4.23 UnpackAll()	442
7.43.4.24 UpdateMouseCursorWhenHover() [1/2]	442
7.43.4.25 UpdateMouseCursorWhenHover() [2/2]	442
7.43.5 Member Data Documentation	443
7.43.5.1 buttons	443
7.43.5.2 isHide	443
7.43.5.3 mChildren	443
7.43.5.4 mIsSelected	443
7.43.5.5 mPosition	444
7.43.5.6 mVisible	444
7.43.5.7 optionContainers	444
7.43.5.8 toggleButton	444
7.44 GUIComponent::OptionInputField Class Reference	444
7.44.1 Detailed Description	447
7.44.2 Member Typedef Documentation	447
7.44.2.1 Ptr	447
7.44.3 Constructor & Destructor Documentation	447
7.44.3.1 OptionInputField()	448
7.44.3.2 ~OptionInputField()	448
7.44.4 Member Function Documentation	448
7.44.4.1 AddInputField()	448
7.44.4.2 AddSubmitField()	448
7.44.4.3 deselect()	448
7.44.4.4 Draw()	448
7.44.4.5 DrawCurrent()	449
7.44.4.6 ExtractInput()	449
7.44.4.7 GetChildren()	449
7.44.4.8 GetHoverStatus() [1/2]	449
7.44.4.9 GetHoverStatus() [2/2]	450
7.44.4.10 GetPosition()	450
7.44.4.11 GetSize()	450
7.44.4.12 GetVisible()	451
7.44.4.13 HasInputField()	451
7.44.4.14 isSelectable()	451
7.44.4.15 isSelected()	451
7.44.4.16 pack()	451
7.44.4.17 select()	452
7.44.4.18 SetNoFieldOption()	452

---

7.44.4.19 SetOption()	452
7.44.4.20 SetPosition() [1/2]	452
7.44.4.21 SetPosition() [2/2]	453
7.44.4.22 SetVisible()	453
7.44.4.23 ToggleInputFields()	453
7.44.4.24 ToggleVisible()	453
7.44.4.25 UnpackAll()	454
7.44.4.26 UpdateMouseCursorWhenHover() [1/2]	454
7.44.4.27 UpdateMouseCursorWhenHover() [2/2]	454
7.44.5 Member Data Documentation	454
7.44.5.1 fonts	455
7.44.5.2 hasInputField	455
7.44.5.3 mChildren	455
7.44.5.4 mInput	455
7.44.5.5 mInputField	455
7.44.5.6 mIsSelected	455
7.44.5.7 mPosition	455
7.44.5.8 mVisible	456
7.45 State::StateStack::PendingChange Struct Reference	456
7.45.1 Detailed Description	456
7.45.2 Constructor & Destructor Documentation	456
7.45.2.1 PendingChange() [1/2]	457
7.45.2.2 PendingChange() [2/2]	457
7.45.3 Member Data Documentation	457
7.45.3.1 action	457
7.45.3.2 stateID	457
7.46 Algorithm::Queue Class Reference	457
7.46.1 Detailed Description	460
7.46.2 Constructor & Destructor Documentation	460
7.46.2.1 Queue() [1/2]	460
7.46.2.2 Queue() [2/2]	460
7.46.2.3 ~Queue()	461
7.46.3 Member Function Documentation	461
7.46.3.1 ApplyInput()	461
7.46.3.2 Dequeue()	461
7.46.3.3 Empty()	461
7.46.3.4 EmptyGenerator()	462
7.46.3.5 Enqueue()	462
7.46.3.6 EnqueueEmpty()	462
7.46.3.7 GenerateAnimation()	462
7.46.3.8 GenerateRelayoutAnimation()	463
7.46.3.9 HighlightArrowFromCur()	463

---

7.46.3.10 InitAction()	464
7.46.3.11 PeekBack()	464
7.46.3.12 PeekFront()	464
7.46.3.13 Random()	464
7.46.3.14 RandomFixedSize()	464
7.46.3.15 RandomFixedSizeGenerator()	465
7.46.3.16 RandomGenerator()	465
7.46.3.17 ReadFromExternalFile()	465
7.46.3.18 ReadFromFileGenerator()	466
7.46.3.19 size()	466
7.46.3.20 UserDefined()	466
7.46.3.21 UserDefinedGenerator()	466
7.46.4 Member Data Documentation	467
7.46.4.1 animController	467
7.46.4.2 codeHighlighter	467
7.46.4.3 maxN	467
7.46.4.4 visualizer	468
7.47 Core::Queue< T > Class Template Reference	468
7.47.1 Detailed Description	472
7.47.2 Member Typedef Documentation	472
7.47.2.1 List	472
7.47.3 Member Function Documentation	472
7.47.3.1 assign() [1/3]	472
7.47.3.2 assign() [2/3]	473
7.47.3.3 assign() [3/3]	473
7.47.3.4 at() [1/2]	473
7.47.3.5 at() [2/2]	474
7.47.3.6 back() [1/2]	474
7.47.3.7 back() [2/2]	474
7.47.3.8 begin() [1/2]	475
7.47.3.9 begin() [2/2]	475
7.47.3.10 clear()	475
7.47.3.11 empty()	475
7.47.3.12 end() [1/2]	475
7.47.3.13 end() [2/2]	475
7.47.3.14 front() [1/2]	476
7.47.3.15 front() [2/2]	476
7.47.3.16 get_iterator()	476
7.47.3.17 insert_previous()	477
7.47.3.18 List() [1/4]	477
7.47.3.19 List() [2/4]	477
7.47.3.20 List() [3/4]	477

---

7.47.3.21 <code>List()</code> [4/4] . . . . .	478
7.47.3.22 <code>move_previous()</code> . . . . .	478
7.47.3.23 <code>operator=()</code> [1/3] . . . . .	478
7.47.3.24 <code>operator=()</code> [2/3] . . . . .	478
7.47.3.25 <code>operator=()</code> [3/3] . . . . .	479
7.47.3.26 <code>operator[]()</code> [1/2] . . . . .	479
7.47.3.27 <code>operator[]()</code> [2/2] . . . . .	479
7.47.3.28 <code>pop()</code> . . . . .	480
7.47.3.29 <code>pop_back()</code> . . . . .	480
7.47.3.30 <code>pop_front()</code> . . . . .	480
7.47.3.31 <code>push()</code> [1/2] . . . . .	480
7.47.3.32 <code>push()</code> [2/2] . . . . .	481
7.47.3.33 <code>push_back()</code> . . . . .	481
7.47.3.34 <code>push_front()</code> . . . . .	481
7.47.3.35 <code>remove()</code> [1/3] . . . . .	482
7.47.3.36 <code>remove()</code> [2/3] . . . . .	482
7.47.3.37 <code>remove()</code> [3/3] . . . . .	482
7.47.3.38 <code>remove_if()</code> [1/2] . . . . .	483
7.47.3.39 <code>remove_if()</code> [2/2] . . . . .	483
7.47.3.40 <code>reset()</code> . . . . .	483
7.47.3.41 <code>resize()</code> [1/2] . . . . .	483
7.47.3.42 <code>resize()</code> [2/2] . . . . .	484
7.47.3.43 <code>reverse()</code> . . . . .	484
7.47.3.44 <code>size()</code> . . . . .	484
7.47.3.45 <code>swap()</code> [1/2] . . . . .	484
7.47.3.46 <code>swap()</code> [2/2] . . . . .	485
7.47.3.47 <code>unique()</code> [1/2] . . . . .	485
7.47.3.48 <code>unique()</code> [2/2] . . . . .	485
7.47.4 Member Data Documentation . . . . .	486
7.47.4.1 <code>mBegin</code> . . . . .	486
7.47.4.2 <code>mEnd</code> . . . . .	486
7.47.4.3 <code>mSize</code> . . . . .	486
7.48 State::QueueState Class Reference . . . . .	487
7.48.1 Detailed Description . . . . .	489
7.48.2 Member Typedef Documentation . . . . .	489
7.48.2.1 <code>Ptr</code> . . . . .	489
7.48.3 Constructor & Destructor Documentation . . . . .	490
7.48.3.1 <code>QueueState()</code> . . . . .	490
7.48.3.2 <code>~QueueState()</code> . . . . .	490
7.48.4 Member Function Documentation . . . . .	490
7.48.4.1 <code>AddDeleteOperation()</code> . . . . .	490
7.48.4.2 <code>AddInitializeOperation()</code> . . . . .	490

---

7.48.4.3 AddInsertOperation()	491
7.48.4.4 AddIntFieldOperationOption()	491
7.48.4.5 AddNoFieldOperationOption()	491
7.48.4.6 AddOperations()	491
7.48.4.7 AddSearchOperation()	491
7.48.4.8 AddStringFieldOption()	492
7.48.4.9 AddUpdateOperation()	492
7.48.4.10 ClearAction()	492
7.48.4.11 ClearError()	492
7.48.4.12 Draw()	492
7.48.4.13 DrawCurrentActionText()	493
7.48.4.14 DrawCurrentErrorText()	493
7.48.4.15 GetContext()	493
7.48.4.16 InitNavigationBar()	493
7.48.4.17 RequestStackClear()	493
7.48.4.18 RequestStackPop()	494
7.48.4.19 RequestStackPush()	494
7.48.4.20 SetCurrentAction()	494
7.48.4.21 SetCurrentError()	494
7.48.4.22 Success()	495
7.48.4.23 Update()	495
7.48.5 Member Data Documentation	495
7.48.5.1 activeDS	495
7.48.5.2 animController	495
7.48.5.3 codeHighlighter	496
7.48.5.4 footer	496
7.48.5.5 mContext	496
7.48.5.6 mCurrentAction	496
7.48.5.7 mCurrentError	496
7.48.5.8 mStack	496
7.48.5.9 navigation	496
7.48.5.10 operationList	497
7.48.5.11 queue	497
7.49 Settings Class Reference	497
7.49.1 Detailed Description	498
7.49.2 Constructor & Destructor Documentation	498
7.49.2.1 Settings() [1/2]	499
7.49.2.2 Settings() [2/2]	499
7.49.2.3 ~Settings()	499
7.49.3 Member Function Documentation	499
7.49.3.1 getColor()	499
7.49.3.2 getInstance()	500

---

7.49.3.3 Load()	500
7.49.3.4 LoadDarkColors()	500
7.49.3.5 LoadDefaultColors()	500
7.49.3.6 operator=()	500
7.49.3.7 SwitchTheme()	501
7.49.4 Member Data Documentation	501
7.49.4.1 mColors	501
7.49.4.2 mCurrentColorTheme	501
7.50 Algorithm::SinglyLinkedList Class Reference	502
7.50.1 Detailed Description	504
7.50.2 Constructor & Destructor Documentation	504
7.50.2.1 SinglyLinkedList() [1/2]	505
7.50.2.2 SinglyLinkedList() [2/2]	505
7.50.2.3 ~SinglyLinkedList()	505
7.50.3 Member Function Documentation	505
7.50.3.1 ApplyInput()	505
7.50.3.2 DeleteHead()	506
7.50.3.3 DeleteMiddle()	506
7.50.3.4 DeleteTail()	506
7.50.3.5 Empty()	506
7.50.3.6 EmptyGenerator()	507
7.50.3.7 GenerateAnimation()	507
7.50.3.8 GenerateRelayoutAnimation()	507
7.50.3.9 HighlightArrowFromCur()	508
7.50.3.10 InitAction()	508
7.50.3.11 InsertAfterTail()	508
7.50.3.12 InsertHead()	509
7.50.3.13 InsertMiddle()	509
7.50.3.14 Random()	509
7.50.3.15 RandomFixedSize()	509
7.50.3.16 RandomFixedSizeGenerator()	510
7.50.3.17 RandomGenerator()	510
7.50.3.18 ReadFromExternalFile()	510
7.50.3.19 ReadFromFileGenerator()	511
7.50.3.20 ResetVisualizer()	511
7.50.3.21 Search()	511
7.50.3.22 size()	512
7.50.3.23 Update()	512
7.50.3.24 UserDefined()	512
7.50.3.25 UserDefinedGenerator()	512
7.50.4 Member Data Documentation	513
7.50.4.1 animController	513

7.50.4.2 codeHighlighter . . . . .	513
7.50.4.3 maxN . . . . .	513
7.50.4.4 visualizer . . . . .	514
7.51 GUIVisualizer::SinglyLinkedList Class Reference . . . . .	514
7.51.1 Detailed Description . . . . .	518
7.51.2 Member Typedef Documentation . . . . .	518
7.51.2.1 Ptr . . . . .	518
7.51.3 Member Enumeration Documentation . . . . .	518
7.51.3.1 ArrowType . . . . .	518
7.51.3.2 Orientation . . . . .	519
7.51.4 Constructor & Destructor Documentation . . . . .	519
7.51.4.1 SinglyLinkedList() [1/2] . . . . .	519
7.51.4.2 SinglyLinkedList() [2/2] . . . . .	519
7.51.4.3 ~SinglyLinkedList() . . . . .	520
7.51.5 Member Function Documentation . . . . .	520
7.51.5.1 DeleteNode() . . . . .	520
7.51.5.2 deselect() . . . . .	520
7.51.5.3 Draw() [1/2] . . . . .	520
7.51.5.4 Draw() [2/2] . . . . .	521
7.51.5.5 DrawArrow() . . . . .	521
7.51.5.6 DrawCurrent() . . . . .	521
7.51.5.7 GenerateNode() . . . . .	522
7.51.5.8 GetArrowType() . . . . .	522
7.51.5.9 GetChildren() . . . . .	522
7.51.5.10 GetHoverStatus() [1/2] . . . . .	522
7.51.5.11 GetHoverStatus() [2/2] . . . . .	522
7.51.5.12 GetList() . . . . .	523
7.51.5.13 GetNodeDefaultPosition() . . . . .	523
7.51.5.14 GetPosition() . . . . .	523
7.51.5.15 GetShape() . . . . .	523
7.51.5.16 GetSize() . . . . .	524
7.51.5.17 GetVisible() . . . . .	524
7.51.5.18 Import() . . . . .	524
7.51.5.19 InsertNode() . . . . .	525
7.51.5.20 isSelectable() . . . . .	525
7.51.5.21 isSelected() . . . . .	525
7.51.5.22 pack() . . . . .	526
7.51.5.23 Relayout() . . . . .	526
7.51.5.24 ResetArrow() . . . . .	526
7.51.5.25 select() . . . . .	526
7.51.5.26 SetArrowType() . . . . .	526
7.51.5.27 SetOrientation() . . . . .	526

---

7.51.5.28 SetPosition() [1/2] . . . . .	527
7.51.5.29 SetPosition() [2/2] . . . . .	527
7.51.5.30 SetShape() . . . . .	527
7.51.5.31 SetShowHeadAndTail() . . . . .	528
7.51.5.32 SetVisible() . . . . .	528
7.51.5.33 size() . . . . .	528
7.51.5.34 ToggleVisible() . . . . .	528
7.51.5.35 UnpackAll() . . . . .	529
7.51.5.36 UpdateMouseCursorWhenHover() [1/2] . . . . .	529
7.51.5.37 UpdateMouseCursorWhenHover() [2/2] . . . . .	529
7.51.6 Member Data Documentation . . . . .	529
7.51.6.1 arrowState . . . . .	530
7.51.6.2 fonts . . . . .	530
7.51.6.3 list . . . . .	530
7.51.6.4 mChildren . . . . .	530
7.51.6.5 mDisplayHeadAndTail . . . . .	530
7.51.6.6 mIsSelected . . . . .	530
7.51.6.7 mNodeDistance . . . . .	530
7.51.6.8 mOrientation . . . . .	531
7.51.6.9 mPosition . . . . .	531
7.51.6.10 mShape . . . . .	531
7.51.6.11 mVisible . . . . .	531
7.52 State::SLLState Class Reference . . . . .	532
7.52.1 Detailed Description . . . . .	534
7.52.2 Member Typedef Documentation . . . . .	534
7.52.2.1 Ptr . . . . .	534
7.52.3 Constructor & Destructor Documentation . . . . .	535
7.52.3.1 SLLState() . . . . .	535
7.52.3.2 ~SLLState() . . . . .	535
7.52.4 Member Function Documentation . . . . .	535
7.52.4.1 AddDeleteOperation() . . . . .	535
7.52.4.2 AddInitializeOperation() . . . . .	535
7.52.4.3 AddInsertOperation() . . . . .	536
7.52.4.4 AddIntFieldOperationOption() . . . . .	536
7.52.4.5 AddNoFieldOperationOption() . . . . .	536
7.52.4.6 AddOperations() . . . . .	536
7.52.4.7 AddSearchOperation() . . . . .	536
7.52.4.8 AddStringFieldOption() . . . . .	537
7.52.4.9 AddUpdateOperation() . . . . .	537
7.52.4.10 ClearAction() . . . . .	537
7.52.4.11 ClearError() . . . . .	537
7.52.4.12 Draw() . . . . .	537

---

7.52.4.13 DrawCurrentActionText()	538
7.52.4.14 DrawCurrentErrorText()	538
7.52.4.15 GetContext()	538
7.52.4.16 InitNavigationBar()	538
7.52.4.17 RequestStackClear()	538
7.52.4.18 RequestStackPop()	539
7.52.4.19 RequestStackPush()	539
7.52.4.20 SetCurrentAction()	539
7.52.4.21 SetCurrentError()	539
7.52.4.22 Success()	540
7.52.4.23 Update()	540
7.52.5 Member Data Documentation	540
7.52.5.1 activeDS	540
7.52.5.2 animController	540
7.52.5.3 codeHighlighter	541
7.52.5.4 footer	541
7.52.5.5 mContext	541
7.52.5.6 mCurrentAction	541
7.52.5.7 mCurrentError	541
7.52.5.8 mStack	541
7.52.5.9 navigation	541
7.52.5.10 operationList	542
7.52.5.11 SLL	542
7.53 Algorithm::Stack Class Reference	542
7.53.1 Detailed Description	545
7.53.2 Constructor & Destructor Documentation	545
7.53.2.1 Stack() [1/2]	545
7.53.2.2 Stack() [2/2]	545
7.53.2.3 ~Stack()	546
7.53.3 Member Function Documentation	546
7.53.3.1 ApplyInput()	546
7.53.3.2 Empty()	546
7.53.3.3 EmptyGenerator()	546
7.53.3.4 GenerateAnimation()	547
7.53.3.5 GenerateRelayoutAnimation()	547
7.53.3.6 HighlightArrowFromCur()	547
7.53.3.7 InitAction()	548
7.53.3.8 Peek()	548
7.53.3.9 Pop()	548
7.53.3.10 Push()	548
7.53.3.11 Random()	549
7.53.3.12 RandomFixedSize()	549

---

7.53.3.13 RandomFixedSizeGenerator()	549
7.53.3.14 RandomGenerator()	550
7.53.3.15 ReadFromExternalFile()	550
7.53.3.16 ReadFromFileGenerator()	550
7.53.3.17 size()	550
7.53.3.18 UserDefined()	551
7.53.3.19 UserDefinedGenerator()	551
7.53.4 Member Data Documentation	551
7.53.4.1 animController	551
7.53.4.2 codeHighlighter	552
7.53.4.3 maxN	552
7.53.4.4 mStackOrientation	552
7.53.4.5 visualizer	552
7.54 Core::Stack< T > Class Template Reference	553
7.54.1 Detailed Description	556
7.54.2 Member Typedef Documentation	556
7.54.2.1 List	556
7.54.3 Member Function Documentation	556
7.54.3.1 assign() [1/3]	557
7.54.3.2 assign() [2/3]	557
7.54.3.3 assign() [3/3]	557
7.54.3.4 at() [1/2]	557
7.54.3.5 at() [2/2]	558
7.54.3.6 back() [1/2]	558
7.54.3.7 back() [2/2]	559
7.54.3.8 begin() [1/2]	559
7.54.3.9 begin() [2/2]	559
7.54.3.10 clear()	559
7.54.3.11 empty()	560
7.54.3.12 end() [1/2]	560
7.54.3.13 end() [2/2]	560
7.54.3.14 front() [1/2]	560
7.54.3.15 front() [2/2]	560
7.54.3.16 get_iterator()	561
7.54.3.17 insert_previous()	561
7.54.3.18 List() [1/4]	562
7.54.3.19 List() [2/4]	562
7.54.3.20 List() [3/4]	562
7.54.3.21 List() [4/4]	562
7.54.3.22 move_previous()	563
7.54.3.23 operator=() [1/3]	563
7.54.3.24 operator=() [2/3]	563

7.54.3.25 operator=() [3/3] . . . . .	563
7.54.3.26 operator[]() [1/2] . . . . .	564
7.54.3.27 operator[]() [2/2] . . . . .	564
7.54.3.28 pop() . . . . .	565
7.54.3.29 pop_back() . . . . .	565
7.54.3.30 pop_front() . . . . .	565
7.54.3.31 push() [1/2] . . . . .	565
7.54.3.32 push() [2/2] . . . . .	566
7.54.3.33 push_back() . . . . .	566
7.54.3.34 push_front() . . . . .	566
7.54.3.35 remove() [1/3] . . . . .	566
7.54.3.36 remove() [2/3] . . . . .	567
7.54.3.37 remove() [3/3] . . . . .	567
7.54.3.38 remove_if() [1/2] . . . . .	568
7.54.3.39 remove_if() [2/2] . . . . .	568
7.54.3.40 reset() . . . . .	568
7.54.3.41 resize() [1/2] . . . . .	568
7.54.3.42 resize() [2/2] . . . . .	569
7.54.3.43 reverse() . . . . .	569
7.54.3.44 size() . . . . .	569
7.54.3.45 swap() [1/2] . . . . .	569
7.54.3.46 swap() [2/2] . . . . .	570
7.54.3.47 top() [1/2] . . . . .	570
7.54.3.48 top() [2/2] . . . . .	570
7.54.3.49 unique() [1/2] . . . . .	570
7.54.3.50 unique() [2/2] . . . . .	571
7.54.4 Member Data Documentation . . . . .	571
7.54.4.1 mBegin . . . . .	571
7.54.4.2 mEnd . . . . .	571
7.54.4.3 mSize . . . . .	571
7.55 State::StackState Class Reference . . . . .	572
7.55.1 Detailed Description . . . . .	574
7.55.2 Member Typedef Documentation . . . . .	574
7.55.2.1 Ptr . . . . .	574
7.55.3 Constructor & Destructor Documentation . . . . .	575
7.55.3.1 StackState() . . . . .	575
7.55.3.2 ~StackState() . . . . .	575
7.55.4 Member Function Documentation . . . . .	575
7.55.4.1 AddDeleteOperation() . . . . .	575
7.55.4.2 AddInitializeOperation() . . . . .	575
7.55.4.3 AddInsertOperation() . . . . .	576
7.55.4.4 AddIntFieldOperationOption() . . . . .	576

---

7.55.4.5 AddNoFieldOperationOption()	576
7.55.4.6 AddOperations()	576
7.55.4.7 AddSearchOperation()	576
7.55.4.8 AddStringFieldOption()	577
7.55.4.9 AddUpdateOperation()	577
7.55.4.10 ClearAction()	577
7.55.4.11 ClearError()	577
7.55.4.12 Draw()	577
7.55.4.13 DrawCurrentActionText()	578
7.55.4.14 DrawCurrentErrorText()	578
7.55.4.15 GetContext()	578
7.55.4.16 InitNavigationBar()	578
7.55.4.17 RequestStackClear()	578
7.55.4.18 RequestStackPop()	579
7.55.4.19 RequestStackPush()	579
7.55.4.20 SetCurrentAction()	579
7.55.4.21 SetCurrentError()	579
7.55.4.22 Success()	580
7.55.4.23 Update()	580
7.55.5 Member Data Documentation	580
7.55.5.1 activeDS	580
7.55.5.2 animController	580
7.55.5.3 codeHighlighter	581
7.55.5.4 footer	581
7.55.5.5 mContext	581
7.55.5.6 mCurrentAction	581
7.55.5.7 mCurrentError	581
7.55.5.8 mStack	581
7.55.5.9 mStackAlgorithm	581
7.55.5.10 navigation	582
7.55.5.11 operationList	582
7.56 State::State Class Reference	582
7.56.1 Detailed Description	583
7.56.2 Member Typedef Documentation	584
7.56.2.1 Ptr	584
7.56.3 Constructor & Destructor Documentation	584
7.56.3.1 State()	584
7.56.3.2 ~State()	584
7.56.4 Member Function Documentation	584
7.56.4.1 Draw()	585
7.56.4.2 GetContext()	585
7.56.4.3 InitNavigationBar()	585

---

7.56.4.4 RequestStackClear()	585
7.56.4.5 RequestStackPop()	586
7.56.4.6 RequestStackPush()	586
7.56.4.7 Update()	586
7.56.5 Member Data Documentation	587
7.56.5.1 mContext	587
7.56.5.2 mStack	587
7.56.5.3 navigation	587
7.57 State::StateStack Class Reference	587
7.57.1 Detailed Description	589
7.57.2 Member Enumeration Documentation	589
7.57.2.1 Action	589
7.57.3 Constructor & Destructor Documentation	590
7.57.3.1 StateStack()	590
7.57.4 Member Function Documentation	590
7.57.4.1 ApplyPendingChanges()	590
7.57.4.2 ClearStates()	590
7.57.4.3 createState()	590
7.57.4.4 Draw()	591
7.57.4.5 IsEmpty()	591
7.57.4.6 PopState()	591
7.57.4.7 PushState()	591
7.57.4.8 RegisterState()	592
7.57.4.9 Update()	592
7.57.5 Member Data Documentation	592
7.57.5.1 mContext	592
7.57.5.2 mFactories	593
7.57.5.3 mPendingList	593
7.57.5.4 mStack	593
7.58 Algorithm::StaticArray Class Reference	593
7.58.1 Detailed Description	596
7.58.2 Constructor & Destructor Documentation	596
7.58.2.1 StaticArray() [1/2]	596
7.58.2.2 StaticArray() [2/2]	596
7.58.2.3 ~StaticArray()	597
7.58.3 Member Function Documentation	597
7.58.3.1 Access()	597
7.58.3.2 ApplyInput()	597
7.58.3.3 Delete()	598
7.58.3.4 Empty()	598
7.58.3.5 EmptyGenerator()	598
7.58.3.6 GenerateAnimation()	598

---

7.58.3.7 GenerateRelayoutAnimation()	599
7.58.3.8 InitAction()	599
7.58.3.9 Insert()	599
7.58.3.10 Random()	600
7.58.3.11 RandomFixedSize()	600
7.58.3.12 RandomFixedSizeGenerator()	600
7.58.3.13 RandomGenerator()	601
7.58.3.14 ReadFromExternalFile()	601
7.58.3.15 ReadFromFileGenerator()	601
7.58.3.16 ResetVisualizer()	602
7.58.3.17 Search()	602
7.58.3.18 size()	602
7.58.3.19 Update()	602
7.58.3.20 UserDefined()	603
7.58.3.21 UserDefinedGenerator()	603
7.58.4 Member Data Documentation	603
7.58.4.1 animController	603
7.58.4.2 codeHighlighter	604
7.58.4.3 maxN	604
7.58.4.4 visualizer	604
7.59 State::StaticArrayState Class Reference	604
7.59.1 Detailed Description	607
7.59.2 Member Typedef Documentation	607
7.59.2.1 Ptr	607
7.59.3 Constructor & Destructor Documentation	607
7.59.3.1 StaticArrayState()	607
7.59.3.2 ~StaticArrayState()	608
7.59.4 Member Function Documentation	608
7.59.4.1 AddAccessOperation()	608
7.59.4.2 AddDeleteOperation()	608
7.59.4.3 AddInitializeOperation()	608
7.59.4.4 AddInsertOperation()	609
7.59.4.5 AddIntFieldOperationOption()	609
7.59.4.6 AddNoFieldOperationOption()	609
7.59.4.7 AddOperations()	609
7.59.4.8 AddSearchOperation()	609
7.59.4.9 AddStringFieldOption()	610
7.59.4.10 AddUpdateOperation()	610
7.59.4.11 ClearAction()	610
7.59.4.12 ClearError()	610
7.59.4.13 Draw()	610
7.59.4.14 DrawCurrentActionText()	611

7.59.4.15 DrawCurrentErrorText()	611
7.59.4.16 GetContext()	611
7.59.4.17 InitNavigationBar()	611
7.59.4.18 RequestStackClear()	611
7.59.4.19 RequestStackPop()	612
7.59.4.20 RequestStackPush()	612
7.59.4.21 SetCurrentAction()	612
7.59.4.22 SetCurrentError()	612
7.59.4.23 Success()	613
7.59.4.24 Update()	613
7.59.5 Member Data Documentation	613
7.59.5.1 activeDS	613
7.59.5.2 animController	613
7.59.5.3 codeHighlighter	614
7.59.5.4 footer	614
7.59.5.5 mContext	614
7.59.5.6 mCurrentAction	614
7.59.5.7 mCurrentError	614
7.59.5.8 mStack	614
7.59.5.9 mStaticArray	614
7.59.5.10 navigation	615
7.59.5.11 operationList	615
7.60 GUIComponent::StringInputField Class Reference	615
7.60.1 Detailed Description	618
7.60.2 Member Typedef Documentation	618
7.60.2.1 Ptr	618
7.60.3 Constructor & Destructor Documentation	618
7.60.3.1 StringInputField()	618
7.60.3.2 ~StringInputField()	619
7.60.4 Member Function Documentation	619
7.60.4.1 AllFieldDisableEdit()	619
7.60.4.2 deselect()	619
7.60.4.3 Draw()	619
7.60.4.4 DrawField()	619
7.60.4.5 ExtractValue()	620
7.60.4.6 GetEditMode()	620
7.60.4.7 GetHoverStatus() [1/2]	620
7.60.4.8 GetHoverStatus() [2/2]	620
7.60.4.9 GetLabel()	621
7.60.4.10 GetPosition()	621
7.60.4.11 GetSize()	621
7.60.4.12 GetVisible()	621

---

7.60.4.13 IsClicked()	622
7.60.4.14 IsSelectable()	622
7.60.4.15 IsSelected()	622
7.60.4.16 Randomize()	622
7.60.4.17 select()	623
7.60.4.18 SetEditMode()	623
7.60.4.19 SetInputFieldSize()	623
7.60.4.20 SetLabel()	623
7.60.4.21 SetLabelSize()	623
7.60.4.22 SetPosition() [1/2]	623
7.60.4.23 SetPosition() [2/2]	624
7.60.4.24 SetVisible()	624
7.60.4.25 ToggleVisible()	624
7.60.4.26 UpdateMouseCursorWhenHover() [1/2]	625
7.60.4.27 UpdateMouseCursorWhenHover() [2/2]	625
7.60.5 Member Data Documentation	625
7.60.5.1 content	625
7.60.5.2 editMode	625
7.60.5.3 extractedValue	626
7.60.5.4 fields	626
7.60.5.5 fonts	626
7.60.5.6 inputFieldSize	626
7.60.5.7 label	626
7.60.5.8 labelFontSize	626
7.60.5.9 mFieldIndex	626
7.60.5.10 mIsSelected	627
7.60.5.11 mMaxLength	627
7.60.5.12 mPosition	627
7.60.5.13 mVisible	627
7.61 TextureHolder Class Reference	628
7.61.1 Detailed Description	629
7.61.2 Constructor & Destructor Documentation	629
7.61.2.1 TextureHolder()	629
7.61.2.2 ~TextureHolder()	629
7.61.3 Member Function Documentation	629
7.61.3.1 Get() [1/2]	629
7.61.3.2 Get() [2/2]	630
7.61.3.3 InsertResource()	630
7.61.3.4 Load() [1/2]	631
7.61.3.5 Load() [2/2]	631
7.61.3.6 LoadFromImage() [1/2]	632
7.61.3.7 LoadFromImage() [2/2]	632

7.61.4 Member Data Documentation . . . . .	633
7.61.4.1 mTextureMap . . . . .	633
7.62 GUIComponent::NavigationBar::TitleInfo Struct Reference . . . . .	633
7.62.1 Member Data Documentation . . . . .	633
7.62.1.1 abbrTitle . . . . .	633
7.62.1.2 stateID . . . . .	633
7.62.1.3 titleName . . . . .	633
<b>8 File Documentation</b> . . . . .	<b>635</b>
8.1 src/Algorithms/Algorithm.hpp File Reference . . . . .	635
8.2 Algorithm.hpp . . . . .	636
8.3 src/Algorithms/Array/DynamicArray.cpp File Reference . . . . .	639
8.4 src/Components/Visualization/DynamicArray.cpp File Reference . . . . .	639
8.5 src/Algorithms/Array/DynamicArray.hpp File Reference . . . . .	640
8.6 DynamicArray.hpp . . . . .	641
8.7 src/Components/Visualization/DynamicArray.hpp File Reference . . . . .	641
8.8 DynamicArray.hpp . . . . .	642
8.9 src/Algorithms/Array/StaticArray.cpp File Reference . . . . .	643
8.10 src/Algorithms/Array/StaticArray.hpp File Reference . . . . .	643
8.11 StaticArray.hpp . . . . .	644
8.12 src/Algorithms/LinkedList/CircularLinkedList.cpp File Reference . . . . .	645
8.12.1 Typedef Documentation . . . . .	646
8.12.1.1 ArrowType . . . . .	646
8.13 src/Components/Visualization/CircularLinkedList.cpp File Reference . . . . .	646
8.14 src/Algorithms/LinkedList/CircularLinkedList.hpp File Reference . . . . .	646
8.15 CircularLinkedList.hpp . . . . .	647
8.16 src/Components/Visualization/CircularLinkedList.hpp File Reference . . . . .	648
8.17 CircularLinkedList.hpp . . . . .	649
8.18 src/Algorithms/LinkedList/DoublyLinkedList.cpp File Reference . . . . .	649
8.18.1 Typedef Documentation . . . . .	650
8.18.1.1 ArrowType . . . . .	650
8.19 src/Components/Visualization/DoublyLinkedList.cpp File Reference . . . . .	650
8.20 src/Algorithms/LinkedList/DoublyLinkedList.hpp File Reference . . . . .	651
8.21 DoublyLinkedList.hpp . . . . .	652
8.22 src/Components/Visualization/DoublyLinkedList.hpp File Reference . . . . .	653
8.23 DoublyLinkedList.hpp . . . . .	653
8.24 src/Algorithms/LinkedList/Queue.cpp File Reference . . . . .	654
8.24.1 Typedef Documentation . . . . .	654
8.24.1.1 ArrowType . . . . .	655
8.25 src/Algorithms/LinkedList/Queue.hpp File Reference . . . . .	655
8.26 Queue.hpp . . . . .	656
8.27 src/Core/Queue.hpp File Reference . . . . .	656

---

8.28 Queue.hpp . . . . .	657
8.29 src/Algorithms/LinkedList/SinglyLinkedList.cpp File Reference . . . . .	657
8.29.1 Typedef Documentation . . . . .	658
8.29.1.1 ArrowType . . . . .	658
8.30 src/Components/Visualization/SinglyLinkedList.cpp File Reference . . . . .	658
8.31 src/Algorithms/LinkedList/SinglyLinkedList.hpp File Reference . . . . .	659
8.32 SinglyLinkedList.hpp . . . . .	660
8.33 src/Components/Visualization/SinglyLinkedList.hpp File Reference . . . . .	661
8.34 SinglyLinkedList.hpp . . . . .	661
8.35 src/Algorithms/LinkedList/Stack.cpp File Reference . . . . .	662
8.36 src/Algorithms/LinkedList/Stack.hpp File Reference . . . . .	662
8.36.1 Typedef Documentation . . . . .	664
8.36.1.1 ArrowType . . . . .	664
8.36.1.2 Orientation . . . . .	664
8.37 Stack.hpp . . . . .	664
8.38 src/Core/Stack.hpp File Reference . . . . .	665
8.39 Stack.hpp . . . . .	665
8.40 src/Animation/AnimationController.hpp File Reference . . . . .	666
8.40.1 Typedef Documentation . . . . .	666
8.40.1.1 CLLAnimationController . . . . .	667
8.40.1.2 DArrayAnimationController . . . . .	667
8.40.1.3 DLLAnimationController . . . . .	667
8.40.1.4 SLLAnimationController . . . . .	667
8.41 AnimationController.hpp . . . . .	667
8.42 src/Animation/AnimationFactory.cpp File Reference . . . . .	670
8.43 src/Animation/AnimationFactory.hpp File Reference . . . . .	671
8.44 AnimationFactory.hpp . . . . .	672
8.45 src/Animation/AnimationState.hpp File Reference . . . . .	672
8.45.1 Typedef Documentation . . . . .	673
8.45.1.1 CLLAnimation . . . . .	674
8.45.1.2 DArrayAnimation . . . . .	674
8.45.1.3 DLLAnimation . . . . .	674
8.45.1.4 SLLAnimation . . . . .	674
8.46 AnimationState.hpp . . . . .	674
8.47 src/Application.cpp File Reference . . . . .	676
8.47.1 Macro Definition Documentation . . . . .	677
8.47.1.1 RAYGUI_IMPLEMENTATION . . . . .	677
8.48 src/Application.hpp File Reference . . . . .	677
8.49 Application.hpp . . . . .	678
8.50 src/Component.cpp File Reference . . . . .	679
8.51 src/Component.hpp File Reference . . . . .	679
8.52 Component.hpp . . . . .	680

---

8.53 src/Components/Common/Button.cpp File Reference . . . . .	681
8.54 src/Components/Common/Button.hpp File Reference . . . . .	681
8.55 Button.hpp . . . . .	682
8.56 src/Components/Common/Card.cpp File Reference . . . . .	683
8.57 src/Components/Common/Card.hpp File Reference . . . . .	683
8.58 Card.hpp . . . . .	684
8.59 src/Components/Common/CodeHighlighter.cpp File Reference . . . . .	685
8.60 src/Components/Common/CodeHighlighter.hpp File Reference . . . . .	685
8.61 CodeHighlighter.hpp . . . . .	686
8.62 src/Components/Common/Footer.hpp File Reference . . . . .	687
8.63 Footer.hpp . . . . .	687
8.64 src/Components/Common/InputField.cpp File Reference . . . . .	689
8.65 src/Components/Common/InputField.hpp File Reference . . . . .	689
8.66 InputField.hpp . . . . .	690
8.67 src/Components/Common/IntegerField.cpp File Reference . . . . .	691
8.68 src/Components/Common/IntegerField.hpp File Reference . . . . .	691
8.69 IntegerInputField.hpp . . . . .	692
8.70 src/Components/Common/NavigationBar.cpp File Reference . . . . .	692
8.71 src/Components/Common/NavigationBar.hpp File Reference . . . . .	692
8.72 NavigationBar.hpp . . . . .	693
8.73 src/Components/Common/OperationContainer.cpp File Reference . . . . .	694
8.74 src/Components/Common/OperationContainer.hpp File Reference . . . . .	694
8.75 OperationContainer.hpp . . . . .	695
8.76 src/Components/Common/OperationList.cpp File Reference . . . . .	696
8.77 src/Components/Common/OperationList.hpp File Reference . . . . .	696
8.78 OperationList.hpp . . . . .	697
8.79 src/Components/Common/OptionInputField.cpp File Reference . . . . .	697
8.80 src/Components/Common/OptionInputField.hpp File Reference . . . . .	698
8.81 OptionInputField.hpp . . . . .	698
8.82 src/Components/Common/StringInputField.cpp File Reference . . . . .	699
8.83 src/Components/Common/StringInputField.hpp File Reference . . . . .	700
8.84 StringInputField.hpp . . . . .	700
8.85 src/Components/Visualization/LinkedList.cpp File Reference . . . . .	701
8.86 src/Components/Visualization/LinkedList.hpp File Reference . . . . .	701
8.87 LinkedList.hpp . . . . .	702
8.88 src/Components/Visualization/Node.cpp File Reference . . . . .	703
8.89 src/Components/Visualization/Node.hpp File Reference . . . . .	703
8.90 Node.hpp . . . . .	704
8.91 src/Core/Node.hpp File Reference . . . . .	705
8.92 Node.hpp . . . . .	706
8.93 src/Container.cpp File Reference . . . . .	706
8.94 src/Container.hpp File Reference . . . . .	707

---

8.95 Container.hpp . . . . .	707
8.96 src/Core/Deque.hpp File Reference . . . . .	708
8.97 Deque.hpp . . . . .	709
8.98 src/Core/List.hpp File Reference . . . . .	709
8.99 List.hpp . . . . .	710
8.100 src/FontHolder.cpp File Reference . . . . .	717
8.101 src/FontHolder.hpp File Reference . . . . .	717
8.102 FontHolder.hpp . . . . .	718
8.103 src/Global.hpp File Reference . . . . .	719
8.104 Global.hpp . . . . .	719
8.105 src/Identifiers/CategoryIdentifiers.hpp File Reference . . . . .	720
8.106 CategoryIdentifiers.hpp . . . . .	720
8.107 src/Identifiers/CategoryInfo.cpp File Reference . . . . .	720
8.108 src/Identifiers/CategoryInfo.hpp File Reference . . . . .	721
8.109 CategoryInfo.hpp . . . . .	721
8.110 src/Identifiers/ColorThemelIdentifiers.hpp File Reference . . . . .	722
8.111 ColorThemelIdentifiers.hpp . . . . .	723
8.112 src/Identifiers/DSIdentifiers.hpp File Reference . . . . .	724
8.113 DSIdentifiers.hpp . . . . .	725
8.114 src/Identifiers/DSInfo.cpp File Reference . . . . .	725
8.115 src/Identifiers/DSInfo.hpp File Reference . . . . .	725
8.116 DSInfo.hpp . . . . .	726
8.117 src/Identifiers/ResourcelIdentifiers.hpp File Reference . . . . .	727
8.118 ResourcelIdentifiers.hpp . . . . .	727
8.119 src/Identifiers/StatelIdentifiers.hpp File Reference . . . . .	728
8.120 StatelIdentifiers.hpp . . . . .	728
8.121 src/Main.cpp File Reference . . . . .	729
8.121.1 Function Documentation . . . . .	729
8.121.1.1 main() . . . . .	729
8.122 src/NonCopyable.hpp File Reference . . . . .	729
8.123 NonCopyable.hpp . . . . .	730
8.124 src/Settings.cpp File Reference . . . . .	730
8.125 src/Settings.hpp File Reference . . . . .	730
8.126 Settings.hpp . . . . .	731
8.127 src/State.cpp File Reference . . . . .	731
8.128 src/State.hpp File Reference . . . . .	732
8.129 State.hpp . . . . .	733
8.130 src/States/Array/ArrayState.hpp File Reference . . . . .	733
8.131 ArrayState.hpp . . . . .	734
8.132 src/States/Array/DynamicArrayState.cpp File Reference . . . . .	738
8.133 src/States/Array/DynamicArrayState.hpp File Reference . . . . .	738
8.134 DynamicArrayState.hpp . . . . .	739

8.135 src/States/Array/StaticArrayState.cpp File Reference	740
8.136 src/States/Array/StaticArrayState.hpp File Reference	740
8.137 StaticArrayState.hpp	741
8.138 src/States/HomepageState.cpp File Reference	742
8.139 src/States/HomepageState.hpp File Reference	742
8.140 HomepageState.hpp	743
8.141 src/States/LinkedList/CLLState.cpp File Reference	743
8.142 src/States/LinkedList/CLLState.hpp File Reference	744
8.143 CLLState.hpp	744
8.144 src/States/LinkedList/DLLState.cpp File Reference	745
8.145 src/States/LinkedList/DLLState.hpp File Reference	745
8.146 DLLState.hpp	746
8.147 src/States/LinkedList/LLState.hpp File Reference	747
8.148 LLState.hpp	748
8.149 src/States/LinkedList/QueueState.cpp File Reference	751
8.150 src/States/LinkedList/QueueState.hpp File Reference	751
8.151 QueueState.hpp	752
8.152 src/States/LinkedList/SLLState.cpp File Reference	753
8.153 src/States/LinkedList/SLLState.hpp File Reference	753
8.154 SLLState.hpp	754
8.155 src/States/LinkedList/StackState.cpp File Reference	755
8.156 src/States/LinkedList/StackState.hpp File Reference	755
8.157 StackState.hpp	756
8.158 src/StateStack.cpp File Reference	756
8.159 src/StateStack.hpp File Reference	757
8.160 StateStack.hpp	758
8.161 src/TextureHolder.cpp File Reference	759
8.162 src/TextureHolder.hpp File Reference	759
8.163 TextureHolder.hpp	760
8.164 src/Utils/Utils.cpp File Reference	760
8.165 src/Utils/Utils.hpp File Reference	761
8.166 Utils.hpp	762
<b>Index</b>	<b>763</b>

# Chapter 1

## Deprecated List

### **Member `GUI::Component::deselect ()`**

This function is deprecated.

### **Member `GUI::Component::isSelectable () const =0`**

This function is deprecated.

### **Member `GUI::Component::isSelected () const`**

This function is deprecated.

### **Member `GUI::Component::mIsSelected`**

This variable is deprecated.

### **Member `GUI::Component::select ()`**

This function is deprecated.

### **Member `GUI::Container::isSelectable () const`**

This function is deprecated.



# Chapter 2

## Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Algorithm . . . . .	15
Animation . . . . .	15
AnimationFactory	
The animation factory namespace, contains all the animation helper functions . . . . .	15
Category . . . . .	18
ColorTheme . . . . .	19
Core . . . . .	21
DataStructures . . . . .	21
Fonts . . . . .	22
global . . . . .	23
GUI . . . . .	23
GUIComponent . . . . .	24
GUIVisualizer . . . . .	24
State . . . . .	25
States . . . . .	25
Textures . . . . .	26
Utils . . . . .	27



# Chapter 3

## Hierarchical Index

### 3.1 Class Hierarchy

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

Algorithm::Algorithm< GUIAlgorithm, AnimationState > . . . . .	29
Algorithm::Algorithm< GUIVisualizer::CircularLinkedList, CLLAnimation > . . . . .	29
Algorithm::CircularLinkedList . . . . .	99
Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList, DLLAnimation > . . . . .	29
Algorithm::DoublyLinkedList . . . . .	208
Algorithm::Algorithm< GUIVisualizer::DynamicArray, DArrayAnimation > . . . . .	29
Algorithm::DynamicArray . . . . .	242
Algorithm::StaticArray . . . . .	593
Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList, SLLAnimation > . . . . .	29
Algorithm::Queue . . . . .	457
Algorithm::SinglyLinkedList . . . . .	502
Algorithm::Stack . . . . .	542
Animation::AnimationController< T > . . . . .	37
Animation::AnimationController< AnimationState > . . . . .	37
Animation::AnimationController< CLLAnimation > . . . . .	37
Animation::AnimationController< DArrayAnimation > . . . . .	37
Animation::AnimationController< DLLAnimation > . . . . .	37
Animation::AnimationController< SLLAnimation > . . . . .	37
Animation::AnimationState< T > . . . . .	47
Application . . . . .	55
GUI::Component . . . . .	151
GUI::Container . . . . .	168
GUIComponent::Footer< CLLAnimationController > . . . . .	286
GUIComponent::Footer< DLLAnimationController > . . . . .	286
GUIComponent::Footer< DArrayAnimationController > . . . . .	286
GUIComponent::Footer< SLLAnimationController > . . . . .	286
GUIComponent::Footer< T > . . . . .	286
GUIComponent::OperationContainer . . . . .	422
GUIComponent::OperationList . . . . .	433
GUIComponent::OptionInputField . . . . .	444
GUIVisualizer::DynamicArray . . . . .	253
GUIVisualizer::LinkedList . . . . .	341
GUIVisualizer::CircularLinkedList . . . . .	111
GUIVisualizer::DoublyLinkedList . . . . .	221

GUIVisualizer::SinglyLinkedList . . . . .	514
GUIComponent::Button . . . . .	71
GUIComponent::Card . . . . .	85
GUIComponent::CodeHighlighter . . . . .	140
GUIComponent::InputField . . . . .	309
GUIComponent::IntegerField . . . . .	322
GUIComponent::StringInputField . . . . .	615
GUIComponent::NavigationBar . . . . .	387
GUIVisualizer::Node . . . . .	402
Core::List< T >::const_iterator . . . . .	160
State::State::Context . . . . .	178
CategoryInfo::Info . . . . .	304
DSInfo::Info . . . . .	306
State::ArrayState< T >::IntegerField . . . . .	320
State::LLState< T >::IntegerField . . . . .	321
Core::List< T >::iterator . . . . .	334
Core::List< T > . . . . .	358
Core::Deque< Component::Ptr > . . . . .	180
Core::Deque< State::StateStack::PendingChange > . . . . .	180
Core::Deque< T > . . . . .	180
Core::Queue< T > . . . . .	468
Core::Stack< T > . . . . .	553
Core::List< Component::Ptr > . . . . .	358
Core::List< State::StateStack::PendingChange > . . . . .	358
Core::Node< T > . . . . .	399
NonCopyable< T > . . . . .	420
NonCopyable< CategoryInfo > . . . . .	420
CategoryInfo . . . . .	96
NonCopyable< DSInfo > . . . . .	420
DSInfo . . . . .	239
NonCopyable< FontHolder > . . . . .	420
FontHolder . . . . .	282
NonCopyable< Settings > . . . . .	420
Settings . . . . .	497
NonCopyable< State > . . . . .	420
State::State . . . . .	582
State::ArrayState< DArrayAnimationController > . . . . .	60
State::DynamicArrayList . . . . .	271
State::StaticArrayList . . . . .	604
State::LLState< CLLAnimationController > . . . . .	375
State::CLLState . . . . .	130
State::LLState< DLLAnimationController > . . . . .	375
State::DLLState . . . . .	197
State::LLState< SLLAnimationController > . . . . .	375
State::QueueState . . . . .	487
State::SLLState . . . . .	532
State::StackState . . . . .	572
State::ArrayList< T > . . . . .	60
State::HomepageState . . . . .	297
State::LLState< T > . . . . .	375
NonCopyable< StateStack > . . . . .	420
State::StateStack . . . . .	587
NonCopyable< TextureHolder > . . . . .	420
TextureHolder . . . . .	628
State::StateStack::PendingChange . . . . .	456
GUIComponent::NavigationBar::TitleInfo . . . . .	633

# Chapter 4

## Class Index

### 4.1 Class List

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

<a href="#">Algorithm::Algorithm&lt; GUIAlgorithm, AnimationState &gt;</a>	
Base class for all algorithms (which is used to generate step-by-step instructions for visualization)	<a href="#">29</a>
<a href="#">Animation::AnimationController&lt; T &gt;</a>	
The animation controller class	<a href="#">37</a>
<a href="#">Animation::AnimationState&lt; T &gt;</a>	
The animation state class	<a href="#">47</a>
<a href="#">Application</a>	
The application class that represents the application	<a href="#">55</a>
<a href="#">State::ArrayState&lt; T &gt;</a>	
The state class that is used as a base array state for all array state/scene of the application	<a href="#">60</a>
<a href="#">GUIComponent::Button</a>	
The button class that is used to represent a button in the <a href="#">GUI</a>	<a href="#">71</a>
<a href="#">GUIComponent::Card</a>	
The card class that is used to represent a card in the <a href="#">GUI</a>	<a href="#">85</a>
<a href="#">CategoryInfo</a>	
The category info class that is used to store information about the categories	<a href="#">96</a>
<a href="#">Algorithm::CircularLinkedList</a>	
The algorithm class that is used to generate step-by-step instructions for the visualization of the circular linked list	<a href="#">99</a>
<a href="#">GUIVisualizer::CircularLinkedList</a>	
The circular linked list visualization	<a href="#">111</a>
<a href="#">State::CLLState</a>	
The state class that is used to represent the circular linked list state/scene of the application	<a href="#">130</a>
<a href="#">GUIComponent::CodeHighlighter</a>	
The code highlighter class that is used to represent a code highlighter in the <a href="#">GUI</a>	<a href="#">140</a>
<a href="#">GUI::Component</a>	
The base component class that is used to represent a <a href="#">GUI</a> component in the <a href="#">GUI</a>	<a href="#">151</a>
<a href="#">Core::List&lt; T &gt;::const_iterator</a>	
The list <code>const_iterator</code> class	<a href="#">160</a>
<a href="#">GUI::Container</a>	
The base container class that is used to represent a <a href="#">GUI</a> container in the <a href="#">GUI</a>	<a href="#">168</a>
<a href="#">State::State::Context</a>	
The context that is used to share the resources (fonts, textures, ...) between states (scenes)	<a href="#">178</a>
<a href="#">Core::Deque&lt; T &gt;</a>	
The deque container	<a href="#">180</a>

<b>State::DLLState</b>	The state that is used to visualize the doubly linked list . . . . .	197
<b>Algorithm::DoublyLinkedList</b>	The algorithm class that is used to generate step-by-step instructions for the visualization of the doubly linked list . . . . .	208
<b>GUIVisualizer::DoublyLinkedList</b>	The doubly linked list visualization . . . . .	221
<b>DSInfo</b>	The data structure info class that is used to store information about the data structures . . . . .	239
<b>Algorithm::DynamicArray</b>	The algorithm class that is used to generate step-by-step instructions for the visualization of the dynamic array . . . . .	242
<b>GUIVisualizer::DynamicArray</b>	The base class for the dynamic array visualization. This class provides the basic functionality for the dynamic array visualization . . . . .	253
<b>State::DynamicArrayList</b>	The state class that is used to represent the dynamic array state/scene of the application . . . . .	271
<b>FontHolder</b>	The font holder class that is used to store the fonts . . . . .	282
<b>GUICOMPONENT::Footer&lt; T &gt;</b>	The footer class that is used to represent a footer in the GUI . . . . .	286
<b>State::HomepageState</b>	The homepage screen . . . . .	297
<b>CategoryInfo::Info</b>	The info struct that is used to store information about the categories . . . . .	304
<b>DSInfo::Info</b>	The info struct that is used to store information about the data structures . . . . .	306
<b>GUICOMPONENT::InputField</b>	The input field class that is used to represent an input field in the GUI . . . . .	309
<b>State::ArrayList&lt; T &gt;::IntegerInput</b>	. . . . .	320
<b>State::LLState&lt; T &gt;::IntegerInput</b>	. . . . .	321
<b>GUICOMPONENT::IntegerInputField</b>	The integer input field class that is used to represent an integer input field in the GUI . . . . .	322
<b>Core::List&lt; T &gt;::iterator</b>	The list iterator class . . . . .	334
<b>GUIVisualizer::LinkedList</b>	The base class for the linked list visualization. This class provides the basic functionality for the linked list visualization . . . . .	341
<b>Core::List&lt; T &gt;</b>	The base container for implementing other data structures . . . . .	358
<b>State::LLState&lt; T &gt;</b>	The state class that is used as a base linked list state/scene of the application . . . . .	375
<b>GUICOMPONENT::NavigationBar</b>	The navigation bar class that is used to represent a navigation bar in the GUI . . . . .	387
<b>Core::Node&lt; T &gt;</b>	The node class that is used to store the value of the node, and the pointers to the previous and the next node (similar to Doubly Linked List node) . . . . .	399
<b>GUIVisualizer::Node</b>	The node class that is used to represent a node in the visualization . . . . .	402
<b>NonCopyable&lt; T &gt;</b>	The self-explanatory non-copyable class . . . . .	420
<b>GUICOMPONENT::OperationContainer</b>	The operation container class that is used to represent an operation container in the GUI . . . . .	422
<b>GUICOMPONENT::OperationList</b>	The operation list class that is used to represent an operation list in the GUI . . . . .	433
<b>GUICOMPONENT::OptionInputField</b>	The option input field class that is used to represent an option input field in the GUI . . . . .	444

<b>State::StateStack::PendingChange</b>	The pending changes that are applied to the state stack . . . . .	456
<b>Algorithm::Queue</b>	The algorithm class that is used to generate step-by-step instructions for the visualization of the queue . . . . .	457
<b>Core::Queue&lt; T &gt;</b>	The queue container . . . . .	468
<b>State::QueueState</b>	The state that is used to visualize the queue . . . . .	487
<b>Settings</b>	The settings class that is used to store the settings . . . . .	497
<b>Algorithm::SinglyLinkedList</b>	The algorithm class that is used to generate step-by-step instructions for the visualization of the singly linked list . . . . .	502
<b>GUIVisualizer::SinglyLinkedList</b>	The singly linked list visualization . . . . .	514
<b>State::SLLState</b>	The state class that is used to represent the singly linked list state/scene of the application . . . . .	532
<b>Algorithm::Stack</b>	The algorithm class that is used to generate step-by-step instructions for the visualization of the stack . . . . .	542
<b>Core::Stack&lt; T &gt;</b>	The stack container . . . . .	553
<b>State::StackState</b>	The state class that is used to represent the stack state/scene of the application . . . . .	572
<b>State::State</b>	The base state class that is used to represent a state/scene of the application . . . . .	582
<b>State::StateStack</b>	The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes) . . . . .	587
<b>Algorithm::StaticArray</b>	The algorithm class that is used to generate step-by-step instructions for the visualization of the static array . . . . .	593
<b>State::StaticArrayList</b>	The state class that is used to represent the static array state/scene of the application . . . . .	604
<b>GUIComponent::StringInputField</b>	The string input field class that is used to represent a string input field in the <b>GUI</b> . . . . .	615
<b>TextureHolder</b>	The image holder class that is used to store the images . . . . .	628
<b>GUIComponent::NavigationBar::TitleInfo</b>	. . . . .	633



# Chapter 5

## File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

src/Application.cpp	676
src/Application.hpp	677
src/Component.cpp	679
src/Component.hpp	679
src/Container.cpp	706
src/Container.hpp	707
src/FontHolder.cpp	717
src/FontHolder.hpp	717
src/Global.hpp	719
src/Main.cpp	729
src/NonCopyable.hpp	729
src/Settings.cpp	730
src/Settings.hpp	730
src/State.cpp	731
src/State.hpp	732
src/StateStack.cpp	756
src/StateStack.hpp	757
src/TextureHolder.cpp	759
src/TextureHolder.hpp	759
src/Algorithms/Algorithm.hpp	635
src/Algorithms/Array/DynamicArray.cpp	639
src/Algorithms/Array/DynamicArray.hpp	640
src/Algorithms/Array/StaticArray.cpp	643
src/Algorithms/Array/StaticArray.hpp	643
src/Algorithms/LinkedList/CircularLinkedList.cpp	645
src/Algorithms/LinkedList/CircularLinkedList.hpp	646
src/Algorithms/LinkedList/DoublyLinkedList.cpp	649
src/Algorithms/LinkedList/DoublyLinkedList.hpp	651
src/Algorithms/LinkedList/Queue.cpp	654
src/Algorithms/LinkedList/Queue.hpp	655
src/Algorithms/LinkedList/SinglyLinkedList.cpp	657
src/Algorithms/LinkedList/SinglyLinkedList.hpp	659
src/Algorithms/LinkedList/Stack.cpp	662
src/Algorithms/LinkedList/Stack.hpp	662
src/Animation/AnimationController.hpp	666

src/Animation/AnimationFactory.cpp	670
src/Animation/AnimationFactory.hpp	671
src/Animation/AnimationState.hpp	672
src/Components/Common/Button.cpp	681
src/Components/Common/Button.hpp	681
src/Components/Common/Card.cpp	683
src/Components/Common/Card.hpp	683
src/Components/Common/CodeHighlighter.cpp	685
src/Components/Common/CodeHighlighter.hpp	685
src/Components/Common/Footer.hpp	687
src/Components/Common/InputField.cpp	689
src/Components/Common/InputField.hpp	689
src/Components/Common/IntegerField.cpp	691
src/Components/Common/IntegerField.hpp	691
src/Components/Common/NavigationBar.cpp	692
src/Components/Common/NavigationBar.hpp	692
src/Components/Common/OperationContainer.cpp	694
src/Components/Common/OperationContainer.hpp	694
src/Components/Common/OperationList.cpp	696
src/Components/Common/OperationList.hpp	696
src/Components/Common/OptionInputField.cpp	697
src/Components/Common/OptionInputField.hpp	698
src/Components/Common/StringInputField.cpp	699
src/Components/Common/StringInputField.hpp	700
src/Components/Visualization/CircularLinkedList.cpp	646
src/Components/Visualization/CircularLinkedList.hpp	648
src/Components/Visualization/DoublyLinkedList.cpp	650
src/Components/Visualization/DoublyLinkedList.hpp	653
src/Components/Visualization/DynamicArray.cpp	639
src/Components/Visualization/DynamicArray.hpp	641
src/Components/Visualization/LinkedList.cpp	701
src/Components/Visualization/LinkedList.hpp	701
src/Components/Visualization/Node.cpp	703
src/Components/Visualization/Node.hpp	703
src/Components/Visualization/SinglyLinkedList.cpp	658
src/Components/Visualization/SinglyLinkedList.hpp	661
src/Core/Deque.hpp	708
src/Core/List.hpp	709
src/Core/Node.hpp	705
src/Core/Queue.hpp	656
src/Core/Stack.hpp	665
src/Identifiers/CategoryIdentifiers.hpp	720
src/Identifiers/CategoryInfo.cpp	720
src/Identifiers/CategoryInfo.hpp	721
src/Identifiers/ColorThemelIdentifiers.hpp	722
src/Identifiers/DSIdentifiers.hpp	724
src/Identifiers/DSInfo.cpp	725
src/Identifiers/DSInfo.hpp	725
src/Identifiers/ResourceIdentifiers.hpp	727
src/Identifiers/StatelIdentifiers.hpp	728
src/States/HomepageState.cpp	742
src/States/HomepageState.hpp	742
src/States/Array/ArrayState.hpp	733
src/States/Array/DynamicArrayState.cpp	738
src/States/Array/DynamicArrayState.hpp	738
src/States/Array/StaticArrayState.cpp	740
src/States/Array/StaticArrayState.hpp	740
src/States/LinkedList/CLLState.cpp	743

src/States/LinkedList/CLLState.hpp	744
src/States/LinkedList/DLLState.cpp	745
src/States/LinkedList/DLLState.hpp	745
src/States/LinkedList/LLState.hpp	747
src/States/LinkedList/QueueState.cpp	751
src/States/LinkedList/QueueState.hpp	751
src/States/LinkedList/SLLState.cpp	753
src/States/LinkedList/SLLState.hpp	753
src/States/LinkedList/StackState.cpp	755
src/States/LinkedList/StackState.hpp	755
src/Utils/Utils.cpp	760
src/Utils/Utils.hpp	761



# Chapter 6

## Namespace Documentation

### 6.1 Algorithm Namespace Reference

#### Classes

- class [Algorithm](#)  
*Base class for all algorithms (which is used to generate step-by-step instructions for visualization)*
- class [CircularLinkedList](#)  
*The algorithm class that is used to generate step-by-step instructions for the visualization of the circular linked list.*
- class [DoublyLinkedList](#)  
*The algorithm class that is used to generate step-by-step instructions for the visualization of the doubly linked list.*
- class [DynamicArray](#)  
*The algorithm class that is used to generate step-by-step instructions for the visualization of the dynamic array.*
- class [Queue](#)  
*The algorithm class that is used to generate step-by-step instructions for the visualization of the queue.*
- class [SinglyLinkedList](#)  
*The algorithm class that is used to generate step-by-step instructions for the visualization of the singly linked list.*
- class [Stack](#)  
*The algorithm class that is used to generate step-by-step instructions for the visualization of the stack.*
- class [StaticArray](#)  
*The algorithm class that is used to generate step-by-step instructions for the visualization of the static array.*

### 6.2 Animation Namespace Reference

#### Classes

- class [AnimationController](#)  
*The animation controller class.*
- class [AnimationState](#)  
*The animation state class.*

### 6.3 AnimationFactory Namespace Reference

The animation factory namespace, contains all the animation helper functions.

## Functions

- float `BounceOut` (float t)
- float `ElasticOut` (float t)
- void `DrawDirectionalArrow` (Vector2 start, Vector2 end, bool active, float t)
- void `DrawActiveArrow` (Vector2 start, Vector2 end, float t)
- void `DrawDoubleDirectionalArrow` (Vector2 start, Vector2 end, bool activeStart, bool activeEnd, float tStart, float tEnd)
- void `DrawDoubleActiveArrow` (Vector2 start, Vector2 end, float tStart, float tEnd)
- void `DrawCircularArrow` (Vector2 start, Vector2 end, bool active, float t)
- float `Dist` (Vector2 p1, Vector2 p2)
- Vector2 `InverseVector` (Vector2 vector)
- Vector2 `MoveNode` (Vector2 src, Vector2 dst, float t)
- Color `BlendColor` (Color src, Color dst, float t)
- void `ReCalculateEnds` (Vector2 &start, Vector2 &end, float radius, bool applyX=true, bool applyY=true)

### 6.3.1 Detailed Description

The animation factory namespace, contains all the animation helper functions.

### 6.3.2 Function Documentation

#### 6.3.2.1 BlendColor()

```
Color AnimationFactory::BlendColor (
    Color src,
    Color dst,
    float t )
```

#### 6.3.2.2 BounceOut()

```
float AnimationFactory::BounceOut (
    float t )
```

#### 6.3.2.3 Dist()

```
float AnimationFactory::Dist (
    Vector2 p1,
    Vector2 p2 )
```

#### 6.3.2.4 DrawActiveArrow()

```
void AnimationFactory::DrawActiveArrow (
    Vector2 start,
    Vector2 end,
    float t )
```

#### 6.3.2.5 DrawCircularArrow()

```
void AnimationFactory::DrawCircularArrow (
    Vector2 start,
    Vector2 end,
    bool active,
    float t )
```

#### 6.3.2.6 DrawDirectionalArrow()

```
void AnimationFactory::DrawDirectionalArrow (
    Vector2 start,
    Vector2 end,
    bool active,
    float t )
```

#### 6.3.2.7 DrawDoubleActiveArrow()

```
void AnimationFactory::DrawDoubleActiveArrow (
    Vector2 start,
    Vector2 end,
    float tStart,
    float tEnd )
```

#### 6.3.2.8 DrawDoubleDirectionalArrow()

```
void AnimationFactory::DrawDoubleDirectionalArrow (
    Vector2 start,
    Vector2 end,
    bool activeStart,
    bool activeEnd,
    float tStart,
    float tEnd )
```

### 6.3.2.9 ElasticOut()

```
float AnimationFactory::ElasticOut (
    float t )
```

### 6.3.2.10 InverseVector()

```
Vector2 AnimationFactory::InverseVector (
    Vector2 vector )
```

### 6.3.2.11 MoveNode()

```
Vector2 AnimationFactory::MoveNode (
    Vector2 src,
    Vector2 dst,
    float t )
```

### 6.3.2.12 ReCalculateEnds()

```
void AnimationFactory::ReCalculateEnds (
    Vector2 & start,
    Vector2 & end,
    float radius,
    bool applyX = true,
    bool applyY = true )
```

## 6.4 Category Namespace Reference

### Enumerations

- enum `ID` { `None` , `Array` , `LinkedList` , `Count` }

### 6.4.1 Enumeration Type Documentation

#### 6.4.1.1 ID

```
enum Category::ID
```

## Enumerator

None	
Array	
LinkedList	
Count	

## 6.5 ColorTheme Namespace Reference

### Enumerations

- enum ID {
 Background , Text , Logo1FirstPart , Logo1SecondPart ,
 Logo2FirstPart , Logo2SecondPart , NavigationBar\_SelectedTitle , NavigationBar\_UnselectedTitle ,
 NavigationBar\_Background , Footer\_Background , Footer\_Icon , Footer\_HoveredIcon ,
 Button\_Background , Button\_HoveredBackground , Button\_Text , Card\_Background ,
 Card\_Text , ActionList\_Text , ActionList\_Background , ActionList\_HoverBackground ,
 InputField\_Inactive , CodeHighlighter\_Background , CodeHighlighter\_Text , CodeHighlighter\_HighlightedLineBackground
 ,
 ActionDescription\_Background , ActionDescription\_Text , Visualizer\_Label , Visualizer\_ErrorText ,
 Visualizer\_ActionText , Visualizer\_Node\_Default\_Outline1 , Visualizer\_Node\_Default\_Outline2 , Visualizer\_Node\_Default\_Background ,
 Visualizer\_Node\_Default\_Background2 , Visualizer\_Node\_Default\_Text1 , Visualizer\_Node\_Default\_Text2 ,
 Visualizer\_Node\_Active\_Outline1 ,
 Visualizer\_Node\_Active\_Outline2 , Visualizer\_Node\_Active\_Background1 , Visualizer\_Node\_Active\_Background2 ,
 Visualizer\_Node\_Active\_Text1 ,
 Visualizer\_Node\_Active\_Text2 , Visualizer\_Node\_ActiveBlue\_Outline1 , Visualizer\_Node\_ActiveBlue\_Outline2 ,
 Visualizer\_Node\_ActiveBlue\_Background1 ,
 Visualizer\_Node\_ActiveBlue\_Background2 , Visualizer\_Node\_ActiveBlue\_Text1 , Visualizer\_Node\_ActiveBlue\_Text2 ,
 Visualizer\_Node\_ActiveGreen\_Outline1 ,
 Visualizer\_Node\_ActiveGreen\_Outline2 , Visualizer\_Node\_ActiveGreen\_Background1 , Visualizer\_Node\_ActiveGreen\_Background2 ,
 Visualizer\_Node\_ActiveGreen\_Text1 ,
 Visualizer\_Node\_ActiveGreen\_Text2 , Visualizer\_Node\_ActiveRed\_Outline1 , Visualizer\_Node\_ActiveRed\_Outline2 ,
 Visualizer\_Node\_ActiveRed\_Background1 ,
 Visualizer\_Node\_ActiveRed\_Background2 , Visualizer\_Node\_ActiveRed\_Text1 , Visualizer\_Node\_ActiveRed\_Text2 ,
 Visualizer\_Node\_Iterated\_Outline1 ,
 Visualizer\_Node\_Iterated\_Outline2 , Visualizer\_Node\_Iterated\_Background1 , Visualizer\_Node\_Iterated\_Background2 ,
 Visualizer\_Node\_Iterated\_Text1 ,
 Visualizer\_Node\_Iterated\_Text2 , Visualizer\_Arrow\_Default , Visualizer\_Arrow\_Active , Count }

#### 6.5.1 Enumeration Type Documentation

##### 6.5.1.1 ID

```
enum ColorTheme::ID
```

## Enumerator

Background	
Text	
Logo1FirstPart	
Logo1SecondPart	
Logo2FirstPart	
Logo2SecondPart	
NavigationBar_SelectedTitle	
NavigationBar_UnselectedTitle	
NavigationBar_Background	
Footer_Background	
Footer_Icon	
Footer_HoveredIcon	
Button_Background	
Button_HoveredBackground	
Button_Text	
Card_Background	
Card_Text	
ActionList_Text	
ActionList_Background	
ActionList_HoverBackground	
InputField_Inactive	
CodeHighlighter_Background	
CodeHighlighter_Text	
CodeHighlighter_HighlightedLineBackground	
ActionDescription_Background	
ActionDescription_Text	
Visualizer_Label	
Visualizer_ErrorText	
Visualizer_ActionText	
Visualizer_Node_Default_Outline1	
Visualizer_Node_Default_Outline2	
Visualizer_Node_Default_Background1	
Visualizer_Node_Default_Background2	
Visualizer_Node_Default_Text1	
Visualizer_Node_Default_Text2	
Visualizer_Node_Active_Outline1	
Visualizer_Node_Active_Outline2	
Visualizer_Node_Active_Background1	
Visualizer_Node_Active_Background2	
Visualizer_Node_Active_Text1	
Visualizer_Node_Active_Text2	
Visualizer_Node_ActiveBlue_Outline1	
Visualizer_Node_ActiveBlue_Outline2	
Visualizer_Node_ActiveBlue_Background1	
Visualizer_Node_ActiveBlue_Background2	
Visualizer_Node_ActiveBlue_Text1	
Visualizer_Node_ActiveBlue_Text2	
Visualizer_Node_ActiveGreen_Outline1	

## Enumerator

Visualizer_Node_ActiveGreen_Outline2	
Visualizer_Node_ActiveGreen_Background1	
Visualizer_Node_ActiveGreen_Background2	
Visualizer_Node_ActiveGreen_Text1	
Visualizer_Node_ActiveGreen_Text2	
Visualizer_Node_ActiveRed_Outline1	
Visualizer_Node_ActiveRed_Outline2	
Visualizer_Node_ActiveRed_Background1	
Visualizer_Node_ActiveRed_Background2	
Visualizer_Node_ActiveRed_Text1	
Visualizer_Node_ActiveRed_Text2	
Visualizer_Node_Iterated_Outline1	
Visualizer_Node_Iterated_Outline2	
Visualizer_Node_Iterated_Background1	
Visualizer_Node_Iterated_Background2	
Visualizer_Node_Iterated_Text1	
Visualizer_Node_Iterated_Text2	
Visualizer_Arrow_Default	
Visualizer_Arrow_Active	
Count	

## 6.6 Core Namespace Reference

### Classes

- class [Deque](#)  
*The deque container.*
- class [List](#)  
*The base container for implementing other data structures.*
- class [Node](#)  
*The node class that is used to store the value of the node, and the pointers to the previous and the next node (similar to Doubly Linked [List](#) node).*
- class [Queue](#)  
*The queue container.*
- class [Stack](#)  
*The stack container.*

## 6.7 DataStructures Namespace Reference

### Enumerations

- enum [ID](#) {  
[None](#) , [StaticArray](#) , [DynamicArray](#) , [SinglyLinkedList](#) ,  
[DoublyLinkedList](#) , [CircularLinkedList](#) , [Stack](#) , [Queue](#) ,  
[Count](#) }

## 6.7.1 Enumeration Type Documentation

### 6.7.1.1 ID

```
enum DataStructures::ID
```

#### Enumerator

None	
StaticArray	
DynamicArray	
SinglyLinkedList	
DoublyLinkedList	
CircularLinkedList	
Stack	
Queue	
Count	

## 6.8 Fonts Namespace Reference

### Enumerations

- enum ID {
 Default , Default\_Italic , Default\_Bold , Silkscreen ,
 Consolas , Courier , Courier\_Bold , Count }

## 6.8.1 Enumeration Type Documentation

### 6.8.1.1 ID

```
enum Fonts::ID
```

#### Enumerator

Default	
Default_Italic	
Default_Bold	
Silkscreen	
Consolas	
Courier	
Courier_Bold	
Count	

## 6.9 global Namespace Reference

### Variables

- `constexpr int SCREEN_WIDTH = 1300`
- `constexpr int SCREEN_HEIGHT = 800`
- `const std::string kTitle = "CS162 - VisuAlgo Clone"`
- `const std::string favicon = "assets/images/favicon.png"`

#### 6.9.1 Variable Documentation

##### 6.9.1.1 favicon

```
const std::string global::favicon = "assets/images/favicon.png"
```

##### 6.9.1.2 kTitle

```
const std::string global::kTitle = "CS162 - VisuAlgo Clone"
```

##### 6.9.1.3 SCREEN\_HEIGHT

```
constexpr int global::SCREEN_HEIGHT = 800 [constexpr]
```

##### 6.9.1.4 SCREEN\_WIDTH

```
constexpr int global::SCREEN_WIDTH = 1300 [constexpr]
```

## 6.10 GUI Namespace Reference

### Classes

- class `Component`  
*The base component class that is used to represent a `GUI` component in the `GUI`.*
- class `Container`  
*The base container class that is used to represent a `GUI` container in the `GUI`.*

## 6.11 GUIComponent Namespace Reference

### Classes

- class [Button](#)  
*The button class that is used to represent a button in the [GUI](#).*
- class [Card](#)  
*The card class that is used to represent a card in the [GUI](#).*
- class [CodeHighlighter](#)  
*The code highlighter class that is used to represent a code highlighter in the [GUI](#).*
- class [Footer](#)  
*The footer class that is used to represent a footer in the [GUI](#).*
- class [InputField](#)  
*The input field class that is used to represent an input field in the [GUI](#).*
- class [IntegerInputField](#)  
*The integer input field class that is used to represent an integer input field in the [GUI](#).*
- class [NavigationBar](#)  
*The navigation bar class that is used to represent a navigation bar in the [GUI](#).*
- class [OperationContainer](#)  
*The operation container class that is used to represent an operation container in the [GUI](#).*
- class [OperationList](#)  
*The operation list class that is used to represent an operation list in the [GUI](#).*
- class [OptionInputField](#)  
*The option input field class that is used to represent an option input field in the [GUI](#).*
- class [StringInputField](#)  
*The string input field class that is used to represent a string input field in the [GUI](#).*

## 6.12 GUIVisualizer Namespace Reference

### Classes

- class [CircularLinkedList](#)  
*The circular linked list visualization.*
- class [DoublyLinkedList](#)  
*The doubly linked list visualization.*
- class [DynamicArray](#)  
*The base class for the dynamic array visualization. This class provides the basic functionality for the dynamic array visualization.*
- class [LinkedList](#)  
*The base class for the linked list visualization. This class provides the basic functionality for the linked list visualization.*
- class [Node](#)  
*The node class that is used to represent a node in the visualization.*
- class [SinglyLinkedList](#)  
*The singly linked list visualization.*

## 6.13 State Namespace Reference

### Classes

- class [ArrayState](#)  
*The state class that is used as a base array state for all array state/scene of the application.*
- class [CLLState](#)  
*The state class that is used to represent the circular linked list state/scene of the application.*
- class [DLLState](#)  
*The state that is used to visualize the doubly linked list.*
- class [DynamicArrayState](#)  
*The state class that is used to represent the dynamic array state/scene of the application.*
- class [HomepageState](#)  
*The homepage screen.*
- class [LLState](#)  
*The state class that is used as a base linked list state/scene of the application.*
- class [QueueState](#)  
*The state that is used to visualize the queue.*
- class [SLLState](#)  
*The state class that is used to represent the singly linked list state/scene of the application.*
- class [StackState](#)  
*The state class that is used to represent the stack state/scene of the application.*
- class [State](#)  
*The base state class that is used to represent a state/scene of the application.*
- class [StateStack](#)  
*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*
- class [StaticArrayState](#)  
*The state class that is used to represent the static array state/scene of the application.*

## 6.14 States Namespace Reference

### Enumerations

- enum [ID](#) {  
  [None](#) , [Homepage](#) , [StaticArray](#) , [DynamicArray](#) ,  
  [SinglyLinkedList](#) , [DoublyLinkedList](#) , [CircularLinkedList](#) , [Stack](#) ,  
  [Queue](#) , [Count](#) }

### 6.14.1 Enumeration Type Documentation

#### 6.14.1.1 ID

```
enum States::ID
```

**Enumerator**

None	
Homepage	
StaticArray	
DynamicArray	
SinglyLinkedList	
DoublyLinkedList	
CircularLinkedList	
Stack	
Queue	
Count	

## 6.15 Textures Namespace Reference

### Enumerations

- enum `ID` {
 `Blank` , `StaticArray` , `DynamicArray` , `SinglyLinkedList` ,
 `DoublyLinkedList` , `CircularLinkedList` , `Stack` , `Queue` ,
 `Favicon` , `Count` }

#### 6.15.1 Enumeration Type Documentation

##### 6.15.1.1 ID

```
enum Textures::ID
```

**Enumerator**

Blank	
StaticArray	
DynamicArray	
SinglyLinkedList	
DoublyLinkedList	
CircularLinkedList	
Stack	
Queue	
Favicon	
Count	

## 6.16 Utils Namespace Reference

### Functions

- std::string [OpenFileDialog](#) (std::string title, std::string description, std::vector< std::string > filters, std::string defaultPath, bool allowMultipleSelect)
- std::string [ReadInputFromFile](#) (std::string path)
- int [Rand](#) (int lower, int upper)
- void [DrawTextBoxed](#) (Font font, const char \*text, Rectangle rec, float fontSize, float spacing, bool wordWrap, Color tint)
- void [DrawTextBoxedSelectable](#) (Font font, const char \*text, Rectangle rec, float fontSize, float spacing, bool wordWrap, Color tint, int selectStart, int selectLength, Color selectTint, Color selectBackTint)
- bool [DrawIcon](#) (int iconID, int x, int y, int pixelSize, Color color, Color hoverColor)

#### 6.16.1 Function Documentation

##### 6.16.1.1 DrawIcon()

```
bool Utils::DrawIcon (
    int iconID,
    int x,
    int y,
    int pixelSize,
    Color color,
    Color hoverColor )
```

##### 6.16.1.2 DrawTextBoxed()

```
void Utils::DrawTextBoxed (
    Font font,
    const char * text,
    Rectangle rec,
    float fontSize,
    float spacing,
    bool wordWrap,
    Color tint )
```

#### 6.16.1.3 DrawTextBboxedSelectable()

```
void Utils::DrawTextBboxedSelectable (
    Font font,
    const char * text,
    Rectangle rec,
    float fontSize,
    float spacing,
    bool wordWrap,
    Color tint,
    int selectStart,
    int selectLength,
    Color selectTint,
    Color selectBackTint )
```

#### 6.16.1.4 OpenFileDialog()

```
std::string Utils::OpenFileDialog (
    std::string title,
    std::string description,
    std::vector< std::string > filters,
    std::string defaultPath,
    bool allowMultipleSelect )
```

#### 6.16.1.5 Rand()

```
int Utils::Rand (
    int lower,
    int upper )
```

#### 6.16.1.6 ReadInputFromFile()

```
std::string Utils::ReadInputFromFile (
    std::string path )
```

# Chapter 7

## Class Documentation

### 7.1 Algorithm::Algorithm< GUIAlgorithm, AnimationState > Class Template Reference

Base class for all algorithms (which is used to generate step-by-step instructions for visualization)

```
#include <Algorithm.hpp>
```

#### Public Member Functions

- **Algorithm (GUIComponent::CodeHighlighter::Ptr codeHighlighter, typename Animation::AnimationController< AnimationState >::Ptr animController, FontHolder \*fonts)**  
*Construct a new Algorithm object.*
- **Algorithm ()**  
*Construct a new Algorithm object.*
- **~Algorithm ()**  
*Destroy the Algorithm object.*
- **virtual void Empty ()**  
*Initialize an empty data structure, and generate animation.*
- **virtual void Random ()**  
*Initialize a data structure with random values input, and then generate animation.*
- **virtual void RandomFixedSize (int N)**  
*Initialize a fixed size data structure with random values input, and then generate animation.*
- **virtual void UserDefined (std::string input)**  
*Initialize a data structure with input from user, and then generate animation.*
- **virtual void ReadFromExternalFile (std::string path)**  
*Initialize a data structure with input from external file (which is used to store the input), and then generate animation.*

## Protected Member Functions

- virtual void [ApplyInput](#) (std::vector< int > input, std::size\_t nMaxSize=10)
 

*Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an std::vector< int >) to the data structure, used by other initialization functions.*
- std::vector< int > [EmptyGenerator](#) ()
 

*Generate an empty input, this is used to generate the input for [Empty\(\)](#)*
- std::vector< int > [RandomGenerator](#) ()
 

*Generate a random input, this is used to generate the input for [Random\(\)](#)*
- std::vector< int > [RandomFixedSizeGenerator](#) (int nSize)
 

*Generate a random input with fixed size, this is used to generate the input for [RandomFixedSize\(\)](#)*
- std::vector< int > [UserDefinedGenerator](#) (std::string input)
 

*Generate an input from user, this is used to generate the input for [UserDefined\(\)](#)*
- std::vector< int > [ReadFromFileGenerator](#) (std::string inputFile)
 

*Parse the input from external file, this is used to generate the input for [ReadFromExternalFile\(\)](#)*
- virtual void [GenerateRelayoutAnimation](#) (Vector2 newPosition)
 

*Generate the relayout animation (which reposition the data structure to a new position on the screen)*
- virtual AnimationState [GenerateAnimation](#) (float duration, int highlightLine, std::string actionDescription)
 

*Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)*
- virtual void [InitAction](#) (std::vector< std::string > code)
 

*Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.*

## Protected Attributes

- GUIAlgorithm [visualizer](#)

*Visualizer for the algorithm (which is used to draw animation generated by the algorithm)*
- GUIComponent::CodeHighlighter::Ptr [codeHighlighter](#)

*Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)*
- Animation::AnimationController< AnimationState >::Ptr [animController](#)

*Animation controller for the algorithm (which is used to control the animation generated by the algorithm)*

### 7.1.1 Detailed Description

```
template<typename GUIAlgorithm, typename AnimationState>
class Algorithm::Algorithm< GUIAlgorithm, AnimationState >
```

Base class for all algorithms (which is used to generate step-by-step instructions for visualization)

#### Template Parameters

<i>GUIAlgorithm</i>	
<i>AnimationState</i>	

### 7.1.2 Constructor & Destructor Documentation

### 7.1.2.1 Algorithm() [1/2]

```
template<typename GUIAlgorithm , typename AnimationState >
Algorithm::Algorithm< GUIAlgorithm, AnimationState >::Algorithm (
    GUIComponent::CodeHighlighter::Ptr codeHighlighter,
    typename Animation::AnimationController< AnimationState >::Ptr animController,
    FontHolder * fonts ) [inline]
```

Construct a new [Algorithm](#) object.

#### Parameters

<i>codeHighlighter</i>	
<i>animController</i>	
<i>fonts</i>	

### 7.1.2.2 Algorithm() [2/2]

```
template<typename GUIAlgorithm , typename AnimationState >
Algorithm::Algorithm< GUIAlgorithm, AnimationState >::Algorithm
```

Construct a new [Algorithm](#) object.

### 7.1.2.3 ~Algorithm()

```
template<typename GUIAlgorithm , typename AnimationState >
Algorithm::Algorithm< GUIAlgorithm, AnimationState >::~Algorithm
```

Destroy the [Algorithm](#) object.

## 7.1.3 Member Function Documentation

### 7.1.3.1 ApplyInput()

```
template<typename GUIAlgorithm , typename AnimationState >
void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::ApplyInput (
    std::vector< int > input,
    std::size_t nMaxSize = 10 ) [protected], [virtual]
```

Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an `std::vector< int >`) to the data structure, used by other initialization functions.

**Parameters**

<i>input</i>	the input (in std::vector< int >) to be applied to the data structure
<i>nMaxSize</i>	the maximum size of the data structure, if the input size is larger than nMaxSize, the input will be truncated to nMaxSize

Reimplemented in [Algorithm::StaticArray](#).

**7.1.3.2 Empty()**

```
template<typename GUIAlgorithm , typename AnimationState >
void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::Empty [virtual]
```

Initialize an empty data structure, and generate animation.

**7.1.3.3 EmptyGenerator()**

```
template<typename GUIAlgorithm , typename AnimationState >
std::vector< int > Algorithm::Algorithm< GUIAlgorithm, AnimationState >::EmptyGenerator [protected]
```

Generate an empty input, this is used to generate the input for [Empty \(\)](#)

**Returns**

std::vector< int > the empty input

**7.1.3.4 GenerateAnimation()**

```
template<typename GUIAlgorithm , typename AnimationState >
AnimationState Algorithm::Algorithm< GUIAlgorithm, AnimationState >::GenerateAnimation (
    float duration,
    int highlightLine,
    std::string actionDescription ) [protected], [virtual]
```

Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)

**Parameters**

<i>duration</i>	the duration of the animation
<i>highlightLine</i>	the line to be highlighted in the code highlighter (if the line is -1, then no line will be highlighted)
<i>actionDescription</i>	the description of the action (which is used to display in the animation)

**Returns**

AnimationState the empty animation state (which currently only contains the metadata of the animation)

**7.1.3.5 GenerateRelayoutAnimation()**

```
template<typename GUIAlgorithm , typename AnimationState >
void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::GenerateRelayoutAnimation (
    Vector2 newPosition ) [inline], [protected], [virtual]
```

Generate the relayout animation (which reposition the data structure to a new position on the screen)

**Parameters**

<i>newPosition</i>	the new position of the data structure
--------------------	--

**7.1.3.6 InitAction()**

```
template<typename GUIAlgorithm , typename AnimationState >
void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::InitAction (
    std::vector< std::string > code ) [protected], [virtual]
```

Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.

**Note**

This function is used to clear the animation controller and the code highlighter, and then initialize a new instruction

**7.1.3.7 Random()**

```
template<typename GUIAlgorithm , typename AnimationState >
void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::Random [virtual]
```

Initialize a data structure with random values input, and then generate animation.

**7.1.3.8 RandomFixedSize()**

```
template<typename GUIAlgorithm , typename AnimationState >
void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::RandomFixedSize (
    int N ) [virtual]
```

Initialize a fixed size data structure with random values input, and then generate animation.

**Parameters**

<b>N</b>	the size of the data structure
----------	--------------------------------

**7.1.3.9 RandomFixedSizeGenerator()**

```
template<typename GUIAlgorithm , typename AnimationState >
std::vector< int > Algorithm::Algorithm< GUIAlgorithm, AnimationState >::RandomFixedSize< Generator (
    int nSize ) [protected]
```

Generate a random input with fixed size, this is used to generate the input for [RandomFixedSize\(\)](#)

**Parameters**

<b>nSize</b>	the size of the input
--------------	-----------------------

**Returns**

`std::vector< int >` the random input with fixed size

**7.1.3.10 RandomGenerator()**

```
template<typename GUIAlgorithm , typename AnimationState >
std::vector< int > Algorithm::Algorithm< GUIAlgorithm, AnimationState >::RandomGenerator
[protected]
```

Generate a random input, this is used to generate the input for [Random\(\)](#)

**Returns**

`std::vector< int >` the random input

**7.1.3.11 ReadFromExternalFile()**

```
template<typename GUIAlgorithm , typename AnimationState >
void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::ReadFromExternalFile (
    std::string path ) [virtual]
```

Initialize a data structure with input from external file (which is used to store the input), and then generate animation.

**Parameters**

<i>path</i>	the path to the external file
-------------	-------------------------------

**7.1.3.12 ReadFromFileGenerator()**

```
template<typename GUIAlgorithm , typename AnimationState >
std::vector< int > Algorithm::Algorithm< GUIAlgorithm, AnimationState >::ReadFromFileGenerator (
    std::string inputFile ) [protected]
```

Parse the input from external file, this is used to generate the input for [ReadFromExternalFile \(\)](#)

**Parameters**

<i>inputFile</i>	the path to the external file
------------------	-------------------------------

**Returns**

`std::vector< int >` the input from external file

**7.1.3.13 UserDefined()**

```
template<typename GUIAlgorithm , typename AnimationState >
void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::UserDefined (
    std::string input ) [virtual]
```

Initialize a data structure with input from user, and then generate animation.

**Parameters**

<i>input</i>	the input from user
--------------	---------------------

**7.1.3.14 UserDefinedGenerator()**

```
template<typename GUIAlgorithm , typename AnimationState >
std::vector< int > Algorithm::Algorithm< GUIAlgorithm, AnimationState >::UserDefinedGenerator (
    std::string input ) [protected]
```

Generate an input from user, this is used to generate the input for [UserDefined \(\)](#)

**Parameters**

<i>input</i>	the input from user
--------------	---------------------

**Returns**

std::vector< int > the input from user

## 7.1.4 Member Data Documentation

### 7.1.4.1 animController

```
template<typename GUIAlgorithm , typename AnimationState >
Animation::AnimationController<AnimationState>::Ptr Algorithm::Algorithm< GUIAlgorithm, AnimationState >::animController [protected]
```

Animation controller for the algorithm (which is used to control the animation generated by the algorithm)

**Note**

This [Algorithm](#) class is mainly use the animation controller to add animation for the algorithm

### 7.1.4.2 codeHighlighter

```
template<typename GUIAlgorithm , typename AnimationState >
GUIComponent::CodeHighlighter::Ptr Algorithm::Algorithm< GUIAlgorithm, AnimationState >::codeHighlighter [protected]
```

Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)

### 7.1.4.3 visualizer

```
template<typename GUIAlgorithm , typename AnimationState >
GUIAlgorithm Algorithm::Algorithm< GUIAlgorithm, AnimationState >::visualizer [protected]
```

Visualizer for the algorithm (which is used to draw animation generated by the algorithm)

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

- src/Algorithms/[Algorithm.hpp](#)

## 7.2 Animation::AnimationController< T > Class Template Reference

The animation controller class.

```
#include <AnimationController.hpp>
```

### Public Types

- `typedef std::shared_ptr< AnimationController > Ptr`  
*The shared pointer of the animation controller.*

### Public Member Functions

- `AnimationController ()`  
*Construct a new Animation Controller object.*
- `~AnimationController ()`  
*Destroy the Animation Controller object.*
- `void RunAll ()`  
*Run all the animations, display the final result.*
- `void Reset ()`  
*Reset the animation controller.*
- `void ResetCurrent ()`  
*Reset the current animation.*
- `void SetAnimation (std::size_t animationIndex)`  
*Set the animation.*
- `std::size_t CurrentAnimationIndex () const`  
*Get the current animation running.*
- `void AddAnimation (T animation)`  
*Add an animation to the end of the animation group.*
- `void PopAnimation ()`  
*remove the last animation from the animation group*
- `void Clear ()`  
*Clear all the animations.*
- `float GetAnimationDuration ()`  
*Get the animation duration.*
- `std::size_t GetNumAnimation () const`  
*Get the number of animations.*
- `std::size_t GetAnimationIndex () const`  
*Get the current animation playing.*
- `bool Done () const`  
*Check if the animation is done.*
- `bool IsPlaying () const`  
*Check if the animation is playing.*
- `void StepForward ()`  
*Move to the next animation.*
- `void StepBackward ()`  
*Move to the previous animation.*
- `void Pause ()`  
*Pause the animation.*

- void **Continue** ()
 

*Continue the animation.*
- void **InteractionLock** ()
 

*Lock the interaction, so the user cannot interact with the animation.*
- void **InteractionAllow** ()
 

*Allow the interaction, so the user can interact with the animation.*
- bool **IsInteractionAllow** () const
 

*Check if the interaction is allowed.*
- void **Update** (float dt)
 

*Update the animation.*
- void **SetSpeed** (float speed)
 

*Adjust the speed of the animation.*
- float **GetSpeed** () const
 

*Get the speed of the animation.*
- T **GetAnimation** ()
 

*Get the current animation playing.*

## Private Member Functions

- float **GetAnimateFrame** (float dt) const
 

*Get the current delta time based on the speed.*
- float **GetStopDuration** ()
 

*Get the current stop duration.*

## Private Attributes

- std::vector< T > **animationGroup**

*The animation group containing all the animation states.*
- std::size\_t **mCurrentAnimationIndex**

*The current animation index.*
- float **mSpeed**

*The current animation.*
- bool **mPlaying**

*The current animation.*
- bool **mInteractionLock**

*The current animation.*
- float **mCurrStopDuration**

*The current delay between two animations.*

## Static Private Attributes

- static constexpr float **defaultSpeed** = 1
 

*The default speed of the animation.*
- static constexpr float **stopDuration** = 0.25
 

*The default delay between two animations.*

### 7.2.1 Detailed Description

```
template<typename T = SLLAnimation>
class Animation::AnimationController< T >
```

The animation controller class.

**Template Parameters**

<i>T</i>	the type of the animation state
----------	---------------------------------

## 7.2.2 Member Typedef Documentation

### 7.2.2.1 Ptr

```
template<typename T = SLLAnimation>
typedef std::shared_ptr< AnimationController > Animation::AnimationController< T >::Ptr
```

The shared pointer of the animation controller.

## 7.2.3 Constructor & Destructor Documentation

### 7.2.3.1 AnimationController()

```
template<typename T >
Animation::AnimationController< T >::AnimationController
```

Construct a new [Animation Controller](#) object.

### 7.2.3.2 ~AnimationController()

```
template<typename T >
Animation::AnimationController< T >::~AnimationController
```

Destroy the [Animation Controller](#) object.

## 7.2.4 Member Function Documentation

### 7.2.4.1 AddAnimation()

```
template<typename T >
void Animation::AnimationController< T >::AddAnimation (
    T animation )
```

Add an animation to the end of the animation group.

**Parameters**

<i>animation</i>	the animation to be added
------------------	---------------------------

**7.2.4.2 Clear()**

```
template<typename T >
void Animation::AnimationController< T >::Clear [inline]
```

Clear all the animations.

**7.2.4.3 Continue()**

```
template<typename T >
void Animation::AnimationController< T >::Continue
```

Continue the animation.

**7.2.4.4 CurrentAnimationIndex()**

```
template<typename T >
std::size_t Animation::AnimationController< T >::CurrentAnimationIndex
```

Get the current animation running.

**Returns**

the current animation running index

**7.2.4.5 Done()**

```
template<typename T >
bool Animation::AnimationController< T >::Done
```

Check if the animation is done.

**Returns**

true if the animation is done, false otherwise

**7.2.4.6 GetAnimateFrame()**

```
template<typename T >
float Animation::AnimationController< T >::GetAnimateFrame (
    float dt ) const [private]
```

Get the current delta time based on the speed.

**Parameters**

<i>dt</i>	the delta time
-----------	----------------

**Returns**

the current delta time based on the speed

**7.2.4.7 GetAnimation()**

```
template<typename T >
T Animation::AnimationController< T >::GetAnimation
```

Get the current animation playing.

**Returns**

the current animation playing

**7.2.4.8 GetAnimationDuration()**

```
template<typename T >
float Animation::AnimationController< T >::GetAnimationDuration
```

Get the animation duration.

**Returns**

the animation duration

**7.2.4.9 GetAnimationIndex()**

```
template<typename T >
std::size_t Animation::AnimationController< T >::GetAnimationIndex
```

Get the current animation playing.

**Returns**

the current animation playing index

#### 7.2.4.10 GetNumAnimation()

```
template<typename T >
std::size_t Animation::AnimationController< T >::GetNumAnimation
```

Get the number of animations.

##### Returns

the number of animations

#### 7.2.4.11 GetSpeed()

```
template<typename T >
float Animation::AnimationController< T >::GetSpeed
```

Get the speed of the animation.

##### Returns

the speed of the animation

#### 7.2.4.12 GetStopDuration()

```
template<typename T >
float Animation::AnimationController< T >::GetStopDuration [inline], [private]
```

Get the current stop duration.

##### Returns

the current stop duration

#### 7.2.4.13 InteractionAllow()

```
template<typename T >
void Animation::AnimationController< T >::InteractionAllow [inline]
```

Allow the interaction, so the user can interact with the animation.

##### Note

By default, the interaction is allowed

#### 7.2.4.14 InteractionLock()

```
template<typename T >
void Animation::AnimationController< T >::InteractionLock [inline]
```

Lock the interaction, so the user cannot interact with the animation.

#### 7.2.4.15 IsInteractionAllow()

```
template<typename T >
bool Animation::AnimationController< T >::IsInteractionAllow [inline]
```

Check if the interaction is allowed.

##### Returns

true if the interaction is allowed, false otherwise

#### 7.2.4.16 IsPlaying()

```
template<typename T >
bool Animation::AnimationController< T >::IsPlaying
```

Check if the animation is playing.

##### Returns

true if the animation is playing, false otherwise

#### 7.2.4.17 Pause()

```
template<typename T >
void Animation::AnimationController< T >::Pause
```

Pause the animation.

#### 7.2.4.18 PopAnimation()

```
template<typename T >
void Animation::AnimationController< T >::PopAnimation
```

remove the last animation from the animation group

#### 7.2.4.19 Reset()

```
template<typename T >
void Animation::AnimationController< T >::Reset
```

Reset the animation controller.

#### 7.2.4.20 ResetCurrent()

```
template<typename T >
void Animation::AnimationController< T >::ResetCurrent
```

Reset the current animation.

#### 7.2.4.21 RunAll()

```
template<typename T >
void Animation::AnimationController< T >::RunAll
```

Run all the animations, display the final result.

#### 7.2.4.22 SetAnimation()

```
template<typename T >
void Animation::AnimationController< T >::SetAnimation (
    std::size_t animationIndex )
```

Set the animation.

##### Parameters

<i>animationIndex</i>	the index of the animation
-----------------------	----------------------------

#### 7.2.4.23 SetSpeed()

```
template<typename T >
void Animation::AnimationController< T >::SetSpeed (
    float speed )
```

Adjust the speed of the animation.

**Parameters**

<i>speed</i>	the speed of the animation
--------------	----------------------------

**7.2.4.24 StepBackward()**

```
template<typename T >
void Animation::AnimationController< T >::StepBackward
```

Move to the previous animation.

**Note**

It will reset the current animation

If the current animation is the first one, it will not move (pretty much the same as reset)

**7.2.4.25 StepForward()**

```
template<typename T >
void Animation::AnimationController< T >::StepForward
```

Move to the next animation.

**Note**

It will reset the current animation

If the current animation is the last one, it will not move (pretty much the same as run all)

**7.2.4.26 Update()**

```
template<typename T >
void Animation::AnimationController< T >::Update (
    float dt )
```

Update the animation.

**Parameters**

<i>dt</i>	the delta time
-----------	----------------

## 7.2.5 Member Data Documentation

### 7.2.5.1 animationGroup

```
template<typename T = SLLAnimation>
std::vector< T > Animation::AnimationController< T >::animationGroup [private]
```

The animation group containing all the animation states.

### 7.2.5.2 defaultSpeed

```
template<typename T = SLLAnimation>
constexpr float Animation::AnimationController< T >::defaultSpeed = 1 [static], [constexpr],
[private]
```

The default speed of the animation.

### 7.2.5.3 mCurrentAnimationIndex

```
template<typename T = SLLAnimation>
std::size_t Animation::AnimationController< T >::mCurrentAnimationIndex [private]
```

The current animation index.

### 7.2.5.4 mCurrStopDuration

```
template<typename T = SLLAnimation>
float Animation::AnimationController< T >::mCurrStopDuration [private]
```

The current delay between two animations.

### 7.2.5.5 mInteractionLock

```
template<typename T = SLLAnimation>
bool Animation::AnimationController< T >::mInteractionLock [private]
```

The current animation.

### 7.2.5.6 mPlaying

```
template<typename T = SLLAnimation>
bool Animation::AnimationController< T >::mPlaying [private]
```

The current animation.

### 7.2.5.7 mSpeed

```
template<typename T = SLLAnimation>
float Animation::AnimationController< T >::mSpeed [private]
```

The current animation.

### 7.2.5.8 stopDuration

```
template<typename T = SLLAnimation>
constexpr float Animation::AnimationController< T >::stopDuration = 0.25 [static], [constexpr],
[private]
```

The default delay between two animations.

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

- src/Animation/AnimationController.hpp

## 7.3 Animation::AnimationState< T > Class Template Reference

The animation state class.

```
#include <AnimationState.hpp>
```

## Public Member Functions

- **AnimationState ()**  
*Construct a new Animation State object.*
- **~AnimationState ()**  
*Destroy the Animation State object.*
- **void PlayingAt (float playingAt)**  
*Move the current playing time to the given time.*
- **float GetCurrentPlayingAt () const**  
*Get the current playing time.*
- **void Draw (Vector2 base=(Vector2){0, 0})**  
*Draw the animation.*
- **void Update (float dt)**  
*Update the current playing time.*
- **void Reset ()**  
*Reset the animation.*
- **void SetDuration (float duration)**  
*Set the duration of the animation.*
- **float GetDuration () const**  
*Get the duration of the animation.*
- **void SetAnimation (std::function< T(T, float, Vector2) > animation)**  
*Set the animation function.*
- **void SetSourceDataStructure (T dataStructure)**  
*Set the source data structure.*
- **T GetDataStructure (float progress, Vector2 base=(Vector2){0, 0})**  
*Get the destination data structure at the given progress.*
- **bool Done () const**  
*Check if the animation is done.*
- **void SetHighlightLine (int line)**  
*Set the highlight line.*
- **int GetHighlightedLine () const**  
*Get the highlighted line.*
- **void SetActionDescription (std::string description)**  
*Set the action description.*
- **std::string GetActionDescription () const**  
*Get the action description.*

## Private Attributes

- **float mDuration**  
*The duration of the animation.*
- **float mCurrentPlayingAt**  
*The current playing time.*
- **int mHighlightedLine**  
*The highlighted line.*
- **std::string actionDescription**  
*The action description.*
- **T mDataStructureBefore**  
*The sourcee data structure of the animation.*
- **std::function< T(T, float, Vector2) > mAnimation**  
*The animation function.*

### 7.3.1 Detailed Description

```
template<typename T>
class Animation::AnimationState< T >
```

The animation state class.

#### Template Parameters

<i>T</i>	the type of the data structure
----------	--------------------------------

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 AnimationState()

```
template<typename T >
Animation::AnimationState< T >::AnimationState
```

Construct a new [Animation State](#) object.

#### 7.3.2.2 ~AnimationState()

```
template<typename T >
Animation::AnimationState< T >::~AnimationState
```

Destroy the [Animation State](#) object.

### 7.3.3 Member Function Documentation

#### 7.3.3.1 Done()

```
template<typename T >
bool Animation::AnimationState< T >::Done
```

Check if the animation is done.

##### Returns

true if the animation is done, false otherwise

#### 7.3.3.2 Draw()

```
template<class T >
void Animation::AnimationState< T >::Draw (
    Vector2 base = (Vector2){0, 0} )
```

Draw the animation.

**Parameters**

<i>base</i>	the base position of the animation
-------------	------------------------------------

**7.3.3.3 GetActionDescription()**

```
template<typename T >
std::string Animation::AnimationState< T >::GetActionDescription [inline]
```

Get the action description.

**Returns**

the action description in std::string type

**7.3.3.4 GetCurrentPlayingAt()**

```
template<typename T >
float Animation::AnimationState< T >::GetCurrentPlayingAt
```

Get the current playing time.

**Returns**

float the current playing time

**7.3.3.5 GetDataStructure()**

```
template<typename T >
T Animation::AnimationState< T >::GetDataStructure (
    float progress,
    Vector2 base = (Vector2){0, 0} )
```

Get the destination data structure at the given progress.

**Parameters**

<i>progress</i>	the progress of the animation
<i>base</i>	the base position of the animation

**Returns**

T the destination data structure

**7.3.3.6 GetDuration()**

```
template<typename T >
float Animation::AnimationState< T >::GetDuration
```

Get the duration of the animation.

**Returns**

float the duration of the animation

**7.3.3.7 GetHighlightedLine()**

```
template<typename T >
int Animation::AnimationState< T >::GetHighlightedLine
```

Get the highlighted line.

**Returns**

int the highlighted line

**7.3.3.8 PlayingAt()**

```
template<typename T >
void Animation::AnimationState< T >::PlayingAt (
    float playingAt )
```

Move the current playing time to the given time.

**Parameters**

<i>playingAt</i>	the time to be moved to
------------------	-------------------------

### 7.3.3.9 Reset()

```
template<typename T >
void Animation::AnimationState< T >::Reset
```

Reset the animation.

### 7.3.3.10 SetActionDescription()

```
template<typename T >
void Animation::AnimationState< T >::SetActionDescription (
    std::string description ) [inline]
```

Set the action description.

#### Parameters

<i>description</i>	the action description
--------------------	------------------------

### 7.3.3.11 SetAnimation()

```
template<typename T >
void Animation::AnimationState< T >::SetAnimation (
    std::function< T(T, float, Vector2) > animation )
```

Set the animation function.

#### Parameters

<i>animation</i>	the animation function
------------------	------------------------

### 7.3.3.12 SetDuration()

```
template<typename T >
void Animation::AnimationState< T >::SetDuration (
    float duration )
```

Set the duration of the animation.

#### Parameters

<i>duration</i>	the duration of the animation
-----------------	-------------------------------

### 7.3.3.13 SetHighlightLine()

```
template<typename T >
void Animation::AnimationState< T >::SetHighlightLine (
    int line )
```

Set the highlight line.

#### Parameters

<i>line</i>	the line to be highlighted
-------------	----------------------------

### 7.3.3.14 SetSourceDataStructure()

```
template<typename T >
void Animation::AnimationState< T >::SetSourceDataStructure (
    T dataStructure )
```

Set the source data structure.

#### Parameters

<i>dataStructure</i>	the source data structure
----------------------	---------------------------

### 7.3.3.15 Update()

```
template<typename T >
void Animation::AnimationState< T >::Update (
    float dt )
```

Update the current playing time.

#### Parameters

<i>dt</i>	the delta time
-----------	----------------

## 7.3.4 Member Data Documentation

#### 7.3.4.1 actionDescription

```
template<typename T >
std::string Animation::AnimationState< T >::actionDescription [private]
```

The action description.

#### 7.3.4.2 mAnimation

```
template<typename T >
std::function< T(T, float, Vector2) > Animation::AnimationState< T >::mAnimation [private]
```

The animation function.

##### Parameters

<i>T</i>	the type of the data structure
<i>float</i>	the progress of the animation
<i>Vector2</i>	the base position of the animation

#### 7.3.4.3 mCurrentPlayingAt

```
template<typename T >
float Animation::AnimationState< T >::mCurrentPlayingAt [private]
```

The current playing time.

#### 7.3.4.4 mDataStructureBefore

```
template<typename T >
T Animation::AnimationState< T >::mDataStructureBefore [private]
```

The sourcee data structure of the animation.

#### 7.3.4.5 mDuration

```
template<typename T >
float Animation::AnimationState< T >::mDuration [private]
```

The duration of the animation.

### 7.3.4.6 mHighlightedLine

```
template<typename T >
int Animation::AnimationState< T >::mHighlightedLine [private]
```

The highlighted line.

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

- src/Animation/AnimationState.hpp

## 7.4 Application Class Reference

The application class that represents the application.

```
#include <Application.hpp>
```

### Public Member Functions

- [Application \(\)](#)  
*Construct a new `Application` object.*
- [~Application \(\)](#)  
*Destroy the `Application` object.*
- void [Run \(\)](#)  
*Run the application.*
- void [Close \(\)](#)  
*Close the application.*
- void [Init \(\)](#)
- bool [WindowClosed \(\)](#)  
*Check if the window is closed.*

### Private Member Functions

- void [Render \(\)](#)  
*Draw/render the application.*
- void [RegisterStates \(\)](#)  
*Register the states/scenes of the application.*
- void [LoadResources \(\)](#)  
*Load the resources of the application.*
- void [Update \(float dt\)](#)  
*Update the application.*

## Private Attributes

- bool [closed](#) = false
- [State::StateStack mStack](#)  
*The state stack of the application.*
- [FontHolder \\* fonts](#)  
*The font holder of the application.*
- [TextureHolder \\* images](#)  
*The texture holder of the application.*
- [CategoryInfo \\* categories](#)  
*The category info of the application.*
- [DSInfo \\* dsInfo](#)  
*The data structure info of the application.*

### 7.4.1 Detailed Description

The application class that represents the application.

The application class is used to represent the application. It contains the main loop of the application.

### 7.4.2 Constructor & Destructor Documentation

#### 7.4.2.1 Application()

```
Application::Application ( )
```

Construct a new [Application](#) object.

#### 7.4.2.2 ~Application()

```
Application::~Application ( )
```

Destroy the [Application](#) object.

### 7.4.3 Member Function Documentation

#### 7.4.3.1 Close()

```
void Application::Close ( )
```

Close the application.

This function closes the application.

#### 7.4.3.2 Init()

```
void Application::Init ( )
```

#### 7.4.3.3 LoadResources()

```
void Application::LoadResources ( ) [private]
```

Load the resources of the application.

This function loads the resources of the application.

##### See also

[TextureHolder::Load](#)

[FontHolder::Load](#)

#### 7.4.3.4 RegisterStates()

```
void Application::RegisterStates ( ) [private]
```

Register the states/scenes of the application.

This function registers the states/scenes of the application.

##### See also

[StateStack::RegisterState](#)

#### 7.4.3.5 Render()

```
void Application::Render ( ) [private]
```

Draw/render the application.

This function draws the application.

#### 7.4.3.6 Run()

```
void Application::Run ( )
```

Run the application.

This function runs the application.

#### 7.4.3.7 Update()

```
void Application::Update ( float dt ) [private]
```

Update the application.

This function updates the application.

**Parameters**

<i>dt</i>	The delta time.
-----------	-----------------

**7.4.3.8 WindowClosed()**

```
bool Application::WindowClosed ( )
```

Check if the window is closed.

This function checks if the window is closed.

**Returns**

True if the window is closed.

**7.4.4 Member Data Documentation****7.4.4.1 categories**

```
CategoryInfo* Application::categories [private]
```

The category info of the application.

The category info is used to hold the category info of the application.

**See also**

[CategoryInfo](#)

**7.4.4.2 closed**

```
bool Application::closed = false [private]
```

#### 7.4.4.3 dsInfo

```
DSInfo* Application::dsInfo [private]
```

The data structure info of the application.

The data structure info is used to hold the all of the information about every data structure.

See also

[DSInfo](#)

#### 7.4.4.4 fonts

```
FontHolder* Application::fonts [private]
```

The font holder of the application.

The font holder is used to hold the fonts of the application.

See also

[FontHolder](#)

#### 7.4.4.5 images

```
TextureHolder* Application::images [private]
```

The texture holder of the application.

The texture holder is used to hold the textures of the application.

See also

[TextureHolder](#)

#### 7.4.4.6 mStack

```
State::StateStack Application::mStack [private]
```

The state stack of the application.

The state stack is used to manage the states/scenes of the application.

See also

[StateStack](#)

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

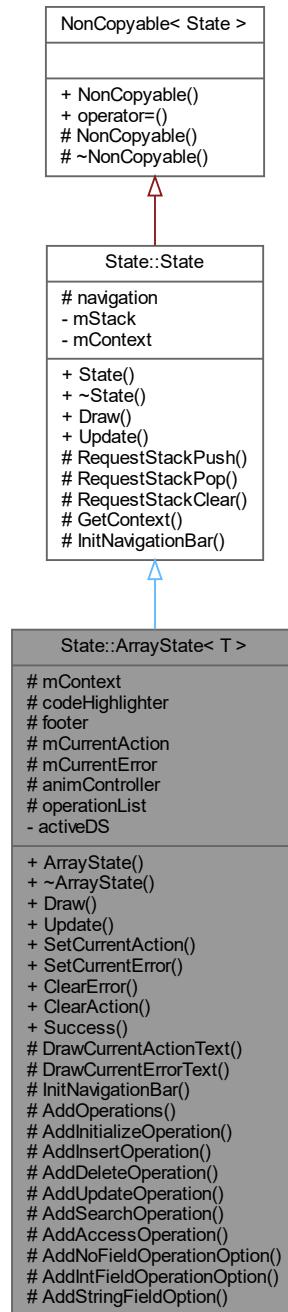
- src/[Application.hpp](#)
- src/[Application.cpp](#)

## 7.5 State::ArrayState< T > Class Template Reference

The state class that is used as a base array state for all array state/scene of the application.

```
#include <ArrayState.hpp>
```

Inheritance diagram for State::ArrayState< T >:



## Classes

- struct [IntegerInput](#)

## Public Types

- `typedef std::unique_ptr< State > Ptr`  
*A unique pointer to the state object.*

## Public Member Functions

- `ArrayState (StateStack &stack, Context context, DataStructures::ID activeDS)`  
*Construct a new `ArrayState` object.*
- `~ArrayState ()`  
*Destroy the `ArrayState` object.*
- `virtual void Draw ()`  
*Draw the array state to the screen.*
- `virtual bool Update (float dt)`  
*Update the array state.*
- `virtual void SetCurrentAction (std::string action)`  
*Set the current action text.*
- `virtual void SetCurrentError (std::string error)`  
*Set the current error text.*
- `virtual void ClearError ()`  
*Clear the current error text.*
- `virtual void ClearAction ()`  
*Clear the current action text.*
- `virtual void Success ()`  
*Clear the current error and toggle the action list.*

## Protected Member Functions

- `virtual void DrawCurrentActionText ()`
- `virtual void DrawCurrentErrorText ()`
- `void InitNavigationBar ()`
- `virtual void AddOperations ()`  
*Add the operations to the operation list.*
- `virtual void AddInitializeOperation ()`  
*Add the initialize operation to the operation list.*
- `virtual void AddInsertOperation ()`  
*Add the insert operation to the operation list.*
- `virtual void AddDeleteOperation ()`  
*Add the delete operation to the operation list.*
- `virtual void AddUpdateOperation ()`  
*Add the update operation to the operation list.*
- `virtual void AddSearchOperation ()`  
*Add the search operation to the operation list.*
- `virtual void AddAccessOperation ()`  
*Add the access operation to the operation list.*
- `virtual void AddNoFieldOperationOption (GUIComponent::OperationContainer::Ptr container, std::string title, std::function< void() > action)`
- `virtual void AddIntFieldOperationOption (GUIComponent::OperationContainer::Ptr container, std::string title, Core::Deque< IntegerInput > fields, std::function< void(std::map< std::string, std::string >) > action)`

- virtual void `AddStringFieldOption (GUIComponent::OperationContainer::Ptr container, std::string title, std::string label, std::function< void(std::map< std::string, std::string >) > action)`
- void `RequestStackPush (States::ID stateID)`  
*Request the state stack to push a new state/scene.*
- void `RequestStackPop ()`  
*Request the state stack to pop the current state/scene.*
- void `RequestStackClear ()`  
*Request the state stack to clear all the states/scenes.*
- `Context GetContext () const`  
*Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).*

## Protected Attributes

- `Context mContext`
- `GUIComponent::CodeHighlighter::Ptr codeHighlighter`
- `GUIComponent::Footer< T > footer`
- `std::string mCurrentAction`
- `std::string mCurrentError`
- `T::Ptr animController`
- `GUIComponent::OperationList operationList`
- `GUIComponent::NavigationBar navigation`

## Private Attributes

- `DataStructures::ID activeDS`
- `StateStack * mStack`

*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*

### 7.5.1 Detailed Description

```
template<typename T>
class State::ArrayState< T >
```

The state class that is used as a base array state for all array state/scene of the application.

### 7.5.2 Member Typedef Documentation

#### 7.5.2.1 Ptr

```
typedef std::unique_ptr< State > State::State::Ptr [inherited]
```

A unique pointer to the state object.

### 7.5.3 Constructor & Destructor Documentation

#### 7.5.3.1 ArrayState()

```
template<typename T >
State::ArrayState< T >::ArrayState (
    StateStack & stack,
    Context context,
    DataStructures::ID activeDS )
```

Construct a new [ArrayState](#) object.

##### Parameters

<code>stack</code>	The state stack where the array state is pushed to.
<code>context</code>	The context of the application.
<code>activeDS</code>	The active data structure of the application.

#### 7.5.3.2 ~ArrayState()

```
template<typename T >
State::ArrayState< T >::~ArrayState
```

Destroy the [ArrayState](#) object.

### 7.5.4 Member Function Documentation

#### 7.5.4.1 AddAccessOperation()

```
template<typename T >
void State::ArrayState< T >::AddAccessOperation [inline], [protected], [virtual]
```

Add the access operation to the operation list.

Reimplemented in [State::DynamicArrayState](#), and [State::StaticArrayState](#).

#### 7.5.4.2 AddDeleteOperation()

```
template<typename T >
void State::ArrayState< T >::AddDeleteOperation [protected], [virtual]
```

Add the delete operation to the operation list.

Reimplemented in [State::DynamicArrayState](#), and [State::StaticArrayState](#).

#### 7.5.4.3 AddInitializeOperation()

```
template<typename T >
void State::ArrayState< T >::AddInitializeOperation [protected], [virtual]
```

Add the initialize operation to the operation list.

Reimplemented in [State::DynamicArrayState](#), and [State::StaticArrayState](#).

#### 7.5.4.4 AddInsertOperation()

```
template<typename T >
void State::ArrayState< T >::AddInsertOperation [protected], [virtual]
```

Add the insert operation to the operation list.

Reimplemented in [State::DynamicArrayState](#), and [State::StaticArrayState](#).

#### 7.5.4.5 AddIntFieldOperationOption()

```
template<typename T >
void State::ArrayState< T >::AddIntFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    Core::Deque< IntegerInput > fields,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual]
```

#### 7.5.4.6 AddNoFieldOperationOption()

```
template<typename T >
void State::ArrayState< T >::AddNoFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::function< void() > action ) [protected], [virtual]
```

#### 7.5.4.7 AddOperations()

```
template<typename T >
void State::ArrayState< T >::AddOperations  [protected], [virtual]
```

Add the operations to the operation list.

##### Note

This function should not be overridden as it calls the other Add\*Operation functions.

#### 7.5.4.8 AddSearchOperation()

```
template<typename T >
void State::ArrayState< T >::AddSearchOperation  [protected], [virtual]
```

Add the search operation to the operation list.

Reimplemented in [State::DynamicArrayList](#), and [State::StaticArrayList](#).

#### 7.5.4.9 AddStringFieldOption()

```
template<typename T >
void State::ArrayState< T >::AddStringFieldOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::string label,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual]
```

#### 7.5.4.10 AddUpdateOperation()

```
template<typename T >
void State::ArrayState< T >::AddUpdateOperation  [protected], [virtual]
```

Add the update operation to the operation list.

Reimplemented in [State::DynamicArrayList](#), and [State::StaticArrayList](#).

#### 7.5.4.11 **ClearAction()**

```
template<typename T >
void State::ArrayState< T >::ClearAction [inline], [virtual]
```

Clear the current action text.

#### 7.5.4.12 **ClearError()**

```
template<typename T >
void State::ArrayState< T >::ClearError [inline], [virtual]
```

Clear the current error text.

#### 7.5.4.13 **Draw()**

```
template<typename T >
void State::ArrayState< T >::Draw [inline], [virtual]
```

Draw the array state to the screen.

Implements [State::State](#).

#### 7.5.4.14 **DrawCurrentActionText()**

```
template<typename T >
void State::ArrayState< T >::DrawCurrentActionText [inline], [protected], [virtual]
```

#### 7.5.4.15 **DrawCurrentErrorText()**

```
template<typename T >
void State::ArrayState< T >::DrawCurrentErrorText [inline], [protected], [virtual]
```

#### 7.5.4.16 GetContext()

```
State::State::Context State::State::GetContext () const [protected], [inherited]
```

Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### Returns

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### See also

[Context](#)

#### 7.5.4.17 InitNavigationBar()

```
template<typename T >
void State::ArrayState< T >::InitNavigationBar [protected]
```

#### 7.5.4.18 RequestStackClear()

```
void State::State::RequestStackClear () [protected], [inherited]
```

Request the state stack to clear all the states/scenes.

##### See also

[StateStack::ClearStates](#)

#### 7.5.4.19 RequestStackPop()

```
void State::State::RequestStackPop () [protected], [inherited]
```

Request the state stack to pop the current state/scene.

##### See also

[StateStack::PopState](#)

#### 7.5.4.20 RequestStackPush()

```
void State::State::RequestStackPush (
    States::ID stateID ) [protected], [inherited]
```

Request the state stack to push a new state/scene.

**Parameters**

<i>stateID</i>	The ID of the state/scene that will be pushed
----------------	---

**See also**

[StateStack::PushState](#)

**7.5.4.21 SetCurrentAction()**

```
template<typename T >
void State::ArrayState< T >::SetCurrentAction (
    std::string action ) [inline], [virtual]
```

Set the current action text.

**Parameters**

<i>action</i>	The current action text.
---------------	--------------------------

**7.5.4.22 SetCurrentError()**

```
template<typename T >
void State::ArrayState< T >::SetCurrentError (
    std::string error ) [inline], [virtual]
```

Set the current error text.

**Parameters**

<i>error</i>	The current error text.
--------------	-------------------------

**7.5.4.23 Success()**

```
template<typename T >
void State::ArrayState< T >::Success [inline], [virtual]
```

Clear the current error and toggle the action list.

#### 7.5.4.24 Update()

```
template<typename T >
bool State::ArrayState< T >::Update (
    float dt ) [virtual]
```

Update the array state.

##### Parameters

<i>dt</i>	The delta time between the previous and the current frame.
-----------	--

##### Returns

true If the update is successful.  
false If the update is unsuccessful.

Implements [State::State](#).

## 7.5.5 Member Data Documentation

### 7.5.5.1 activeDS

```
template<typename T >
DataStructures::ID State::ArrayState< T >::activeDS [private]
```

### 7.5.5.2 animController

```
template<typename T >
T::Ptr State::ArrayState< T >::animController [protected]
```

### 7.5.5.3 codeHighlighter

```
template<typename T >
GUIComponent::CodeHighlighter::Ptr State::ArrayState< T >::codeHighlighter [protected]
```

#### 7.5.5.4 footer

```
template<typename T >
GUIComponent::Footer< T > State::ArrayState< T >::footer [protected]
```

#### 7.5.5.5 mContext

```
template<typename T >
Context State::ArrayState< T >::mContext [protected]
```

#### 7.5.5.6 mCurrentAction

```
template<typename T >
std::string State::ArrayState< T >::mCurrentAction [protected]
```

#### 7.5.5.7 mCurrentError

```
template<typename T >
std::string State::ArrayState< T >::mCurrentError [protected]
```

#### 7.5.5.8 mStack

```
StateStack* State::State::mStack [private], [inherited]
```

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

#### 7.5.5.9 navigation

```
GUIComponent::NavigationBar State::State::navigation [protected], [inherited]
```

### 7.5.5.10 operationList

```
template<typename T >
GUIComponent::OperationList State::ArrayState< T >::operationList [protected]
```

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

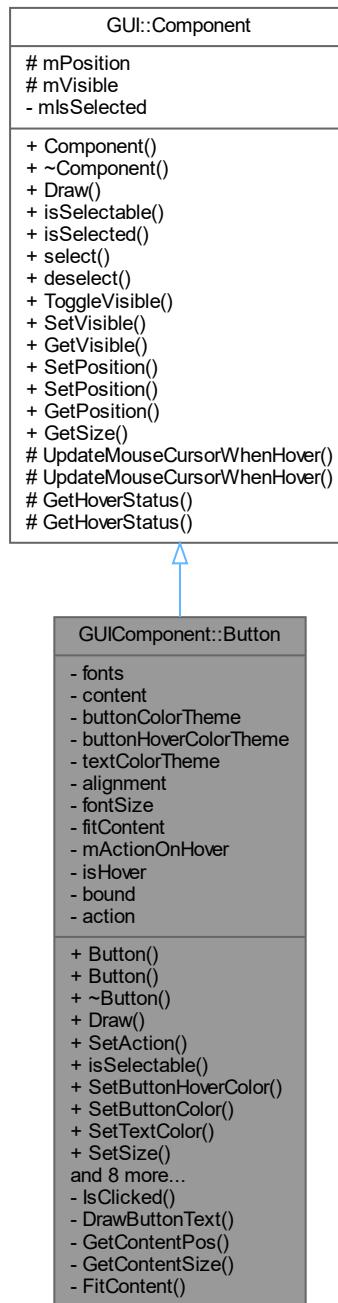
- src/States/Array/[ArrayState.hpp](#)

## 7.6 GUIComponent::Button Class Reference

The button class that is used to represent a button in the [GUI](#).

```
#include <Button.hpp>
```

Inheritance diagram for GUIComponent::Button:



## Public Types

- enum `TextAlignment` { `Left` , `Center` , `Right` , `AlignmentCount` }
- typedef `std::shared_ptr<Button>` `Ptr`

## Public Member Functions

- `Button (std::string text, FontHolder *fonts)`

- `Button ()`
- `~Button ()`
- void `Draw` (Vector2 base=(Vector2){0, 0})  
*Draw the component.*
- void `SetAction` (std::function< void() > clickedAction)
- bool `IsSelectable () const`  
*Check if the component is selectable.*
- void `SetButtonHoverColor` (ColorTheme::ID color)
- void `SetButtonColor` (ColorTheme::ID color)
- void `SetTextColor` (ColorTheme::ID color)
- void `SetSize` (float width, float height)
- void `SetText` (std::string text)
- void `SetFontStyle` (float textFontSize)
- float `GetFontSize () const`
- void `SetTextAlignment` (TextAlignment textAlignment)
- void `EnableFitContent ()`
- void `DisableFitContent ()`
- Vector2 `GetSize ()`  
*Get the size of the component.*
- void `SetActionOnHover` (bool actionOnHover)
- bool `isSelected () const`  
*Check if the component is selected.*
- virtual void `select ()`  
*Select the component.*
- virtual void `deselect ()`  
*Deselect the component.*
- virtual void `ToggleVisible ()`  
*Toggle the visibility of the component.*
- virtual void `SetVisible` (bool visible)  
*Set the visibility of the component.*
- virtual bool `GetVisible ()`  
*Get the visibility of the component.*
- void `SetPosition` (float x, float y)  
*Set the position of the component.*
- void `SetPosition` (Vector2 position)  
*Set the position of the component.*
- Vector2 `GetPosition ()`  
*Get the position of the component.*

## Protected Member Functions

- virtual void `UpdateMouseCursorWhenHover` (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)  
*Update the mouse cursor when hovering.*
- virtual void `UpdateMouseCursorWhenHover` (Rectangle bound, bool hover, bool noHover)  
*Update the mouse cursor when hovering.*
- virtual bool `GetHoverStatus` (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)  
*Get the hover status of the component.*
- virtual bool `GetHoverStatus` (Rectangle bound, bool hover, bool noHover)  
*Get the hover status of the component.*

## Protected Attributes

- Vector2 `mPosition`  
*The position of the component.*
- bool `mVisible`  
*The visibility of the component.*

## Private Member Functions

- bool `IsClicked ()`
- void `DrawButtonText ()`
- Vector2 `GetContentPos ()`
- Vector2 `GetContentSize ()`
- void `FitContent ()`

## Private Attributes

- `FontHolder * fonts`
- std::string `content`
- `ColorTheme::ID buttonColorTheme`
- `ColorTheme::ID buttonHoverColorTheme`
- `ColorTheme::ID textColorTheme`
- `TextAlignment alignment`
- float `fontSize`
- bool `fitContent`
- bool `mActionOnHover`
- bool `isHover`
- Rectangle `bound`
- std::function< void() > `action`
- bool `mlsSelected`

*The selected status of the component.*

### 7.6.1 Detailed Description

The button class that is used to represent a button in the [GUI](#).

### 7.6.2 Member Typedef Documentation

#### 7.6.2.1 Ptr

```
typedef std::shared_ptr< Button > GUIComponent::Button::Ptr
```

### 7.6.3 Member Enumeration Documentation

#### 7.6.3.1 TextAlignment

```
enum GUIComponent::Button::TextAlignment
```

## Enumerator

Left	
Center	
Right	
AlignmentCount	

## 7.6.4 Constructor & Destructor Documentation

### 7.6.4.1 Button() [1/2]

```
GUIComponent::Button::Button (
    std::string text,
    FontHolder * fonts )
```

### 7.6.4.2 Button() [2/2]

```
GUIComponent::Button::Button ( )
```

### 7.6.4.3 ~Button()

```
GUIComponent::Button::~Button ( )
```

## 7.6.5 Member Function Documentation

### 7.6.5.1 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

### 7.6.5.2 DisableFitContent()

```
void GUIComponent::Button::DisableFitContent ( )
```

### 7.6.5.3 Draw()

```
void GUIComponent::Button::Draw (   
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Draw the component.

This function draws the component.

**Parameters**

<i>base</i>	The base position of the component.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

**7.6.5.4 DrawButtonText()**

```
void GUIComponent::Button::DrawButtonText ( ) [private]
```

**7.6.5.5 EnableFitContent()**

```
void GUIComponent::Button::EnableFitContent ( )
```

**7.6.5.6 FitContent()**

```
void GUIComponent::Button::FitContent ( ) [private]
```

**7.6.5.7 GetContentPos()**

```
Vector2 GUIComponent::Button::GetContentPos ( ) [private]
```

**7.6.5.8 GetContentSize()**

```
Vector2 GUIComponent::Button::GetContentSize ( ) [private]
```

**7.6.5.9 GetFontSize()**

```
float GUIComponent::Button::GetFontSize ( ) const
```

**7.6.5.10 GetHoverStatus() [1/2]**

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

**7.6.5.11 GetHoverStatus() [2/2]**

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.6.5.12 GetPosition()**

```
Vector2 GUI::Component::GetPosition () [inherited]
```

Get the position of the component.

This function gets the position of the component.

**Returns**

The position of the component.

**7.6.5.13 GetSize()**

```
Vector2 GUIComponent::Button::GetSize () [virtual]
```

Get the size of the component.

This function gets the size of the component.

**Returns**

The size of the component.

Reimplemented from [GUI::Component](#).

#### 7.6.5.14 GetVisible()

```
bool GUI::Component::GetVisible () [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

##### Returns

The visibility of the component.

#### 7.6.5.15 IsClicked()

```
bool GUIComponent::Button::IsClicked () [private]
```

#### 7.6.5.16 isSelectable()

```
bool GUIComponent::Button::isSelectable () const [virtual]
```

Check if the component is selectable.

**Deprecated** This function is deprecated.

This function checks if the component is selectable.

##### Returns

True if the component is selectable.

Implements [GUI::Component](#).

#### 7.6.5.17 isSelected()

```
bool GUI::Component::isSelected () const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.6.5.18 select()

```
void GUI::Component::select () [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

#### 7.6.5.19 SetAction()

```
void GUIComponent::Button::SetAction (
    std::function< void() > clickedAction )
```

#### 7.6.5.20 SetActionOnHover()

```
void GUIComponent::Button::SetActionOnHover (
    bool actionOnHover )
```

#### 7.6.5.21 SetButtonColor()

```
void GUIComponent::Button::SetButtonColor (
    ColorTheme::ID color )
```

#### 7.6.5.22 SetButtonHoverColor()

```
void GUIComponent::Button::SetButtonHoverColor (
    ColorTheme::ID color )
```

#### 7.6.5.23 SetFontSize()

```
void GUIComponent::Button::SetFontSize (
    float textSize )
```

#### 7.6.5.24 SetPosition() [1/2]

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

**7.6.5.25 SetPosition() [2/2]**

```
void GUI::Component::SetPosition (
    Vector2 position ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>position</i>	The position of the component.
-----------------	--------------------------------

**7.6.5.26 SetSize()**

```
void GUIComponent::Button::SetSize (
    float width,
    float height )
```

**7.6.5.27 SetText()**

```
void GUIComponent::Button::SetText (
    std::string text )
```

**7.6.5.28 SetTextAlignment()**

```
void GUIComponent::Button::SetTextAlignment (
    TextAlignment textAlignment )
```

### 7.6.5.29 SetTextColor()

```
void GUIComponent::Button::SetTextColor (
    ColorTheme::ID color )
```

### 7.6.5.30 SetVisible()

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

#### Parameters

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

### 7.6.5.31 ToggleVisible()

```
void GUI::Component::ToggleVisible () [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

### 7.6.5.32 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

#### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

### 7.6.5.33 UpdateMouseCursorWhenHover() [2/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

#### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

## 7.6.6 Member Data Documentation

### 7.6.6.1 action

```
std::function< void() > GUIComponent::Button::action [private]
```

### 7.6.6.2 alignment

```
TextAlignment GUIComponent::Button::alignment [private]
```

### 7.6.6.3 bound

```
Rectangle GUIComponent::Button::bound [private]
```

### 7.6.6.4 buttonColorTheme

```
ColorTheme::ID GUIComponent::Button::buttonColorTheme [private]
```

### 7.6.6.5 buttonHoverColorTheme

```
ColorTheme::ID GUIComponent::Button::buttonHoverColorTheme [private]
```

### 7.6.6.6 content

```
std::string GUIComponent::Button::content [private]
```

### 7.6.6.7 fitContent

```
bool GUIComponent::Button::fitContent [private]
```

### 7.6.6.8 fonts

```
FontHolder* GUIComponent::Button::fonts [private]
```

### 7.6.6.9 fontSize

```
float GUIComponent::Button::fontSize [private]
```

### 7.6.6.10 isHover

```
bool GUIComponent::Button::isHover [private]
```

### 7.6.6.11 mActionOnHover

```
bool GUIComponent::Button::mActionOnHover [private]
```

### 7.6.6.12 mIsSelected

```
bool GUIComponent::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

### 7.6.6.13 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

### 7.6.6.14 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

### 7.6.6.15 textColorTheme

```
ColorTheme::ID GUIComponent::Button::textColorTheme [private]
```

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

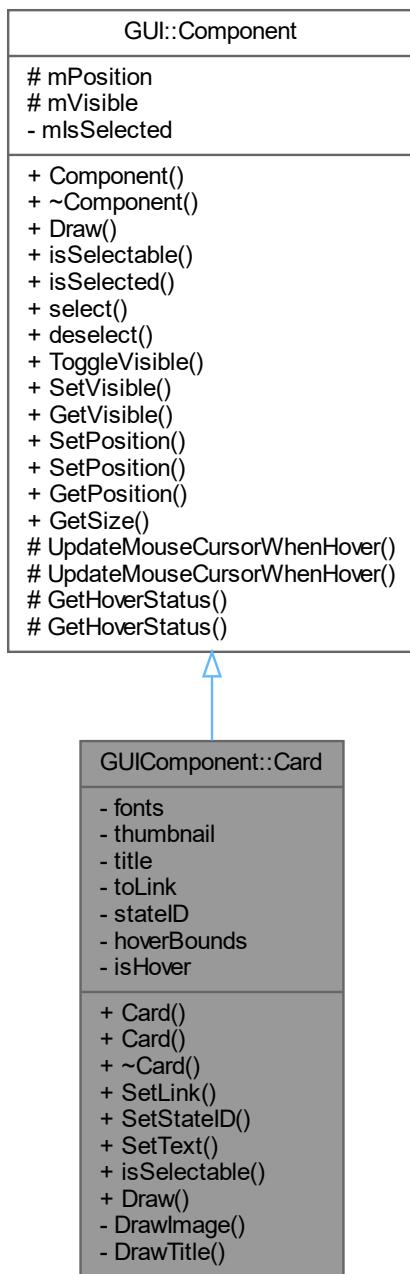
- src/Components/Common/Button.hpp
- src/Components/Common/Button.cpp

## 7.7 GUIComponent::Card Class Reference

The card class that is used to represent a card in the [GUI](#).

```
#include <Card.hpp>
```

Inheritance diagram for GUIComponent::Card:



## Public Types

- `typedef std::shared_ptr< Component > Ptr`  
*The shared pointer to the component.*

## Public Member Functions

- `Card` (std::string text, Texture thumbnail, FontHolder \*fonts)
- `Card` ()
- `~Card` ()
- void `SetLink` (std::function< void(States::ID) > link)
- void `SetStateID` (States::ID stateID)
- void `SetText` (std::string text)
- bool `isSelectable` () const

*Check if the component is selectable.*
- void `Draw` (Vector2 base=(Vector2){0, 0})

*Draw the component.*
- bool `isSelected` () const

*Check if the component is selected.*
- virtual void `select` ()

*Select the component.*
- virtual void `deselect` ()

*Deselect the component.*
- virtual void `ToggleVisible` ()

*Toggle the visibility of the component.*
- virtual void `SetVisible` (bool visible)

*Set the visibility of the component.*
- virtual bool `GetVisible` ()

*Get the visibility of the component.*
- void `SetPosition` (float x, float y)

*Set the position of the component.*
- void `SetPosition` (Vector2 position)

*Set the position of the component.*
- Vector2 `GetPosition` ()

*Get the position of the component.*
- virtual Vector2 `GetSize` ()

*Get the size of the component.*

## Protected Member Functions

- virtual void `UpdateMouseCursorWhenHover` (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)

*Update the mouse cursor when hovering.*
- virtual void `UpdateMouseCursorWhenHover` (Rectangle bound, bool hover, bool noHover)

*Update the mouse cursor when hovering.*
- virtual bool `GetHoverStatus` (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)

*Get the hover status of the component.*
- virtual bool `GetHoverStatus` (Rectangle bound, bool hover, bool noHover)

*Get the hover status of the component.*

## Protected Attributes

- Vector2 `mPosition`

*The position of the component.*
- bool `mVisible`

*The visibility of the component.*

## Private Member Functions

- bool `DrawImage` (Vector2 base)
- bool `DrawTitle` (Vector2 base)

## Private Attributes

- `FontHolder * fonts`
- Texture `thumbnail`
- std::string `title`
- std::function< void(`States::ID`) > `toLink`
- `States::ID stateID`
- std::map< std::string, Rectangle > `hoverBounds`
- bool `isHover`
- bool `isSelected`

*The selected status of the component.*

### 7.7.1 Detailed Description

The card class that is used to represent a card in the [GUI](#).

### 7.7.2 Member Typedef Documentation

#### 7.7.2.1 Ptr

```
typedef std::shared_ptr< Component > GUI::Component::Ptr [inherited]
```

The shared pointer to the component.

The [GUI](#) component is commonly used by multiple components, so a shared pointer is used.

See also

```
std::shared_ptr
```

### 7.7.3 Constructor & Destructor Documentation

#### 7.7.3.1 Card() [1/2]

```
GUIComponent::Card::Card (
    std::string text,
    Texture thumbnail,
    FontHolder * fonts )
```

### 7.7.3.2 Card() [2/2]

```
GUIComponent::Card::Card ( )
```

### 7.7.3.3 ~Card()

```
GUIComponent::Card::~Card ( )
```

## 7.7.4 Member Function Documentation

### 7.7.4.1 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

### 7.7.4.2 Draw()

```
void GUIComponent::Card::Draw ( 
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Draw the component.

This function draws the component.

#### Parameters

<i>base</i>	The base position of the component.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

### 7.7.4.3 DrawImage()

```
bool GUIComponent::Card::DrawImage ( 
    Vector2 base ) [private]
```

#### 7.7.4.4 DrawTitle()

```
bool GUIComponent::Card::DrawTitle (
    Vector2 base ) [private]
```

#### 7.7.4.5 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.7.4.6 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

#### 7.7.4.7 GetPosition()

```
Vector2 GUI::Component::GetPosition () [inherited]
```

Get the position of the component.

This function gets the position of the component.

**Returns**

The position of the component.

#### 7.7.4.8 GetSize()

```
Vector2 GUI::Component::GetSize () [virtual], [inherited]
```

Get the size of the component.

This function gets the size of the component.

**Returns**

The size of the component.

Reimplemented in [GUIComponent::Button](#), [GUIComponent::InputField](#), [GUIComponent::OperationList](#), [GUIComponent::OptionInputField](#) and [GUI::Container](#).

#### 7.7.4.9 GetVisible()

```
bool GUI::Component::GetVisible () [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

**Returns**

The visibility of the component.

#### 7.7.4.10 isSelectable()

```
bool GUIComponent::Card::isSelectable () const [virtual]
```

Check if the component is selectable.

**Deprecated** This function is deprecated.

This function checks if the component is selectable.

**Returns**

True if the component is selectable.

Implements [GUI::Component](#).

#### 7.7.4.11 isSelected()

```
bool GUI::Component::isSelected ( ) const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.7.4.12 select()

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

#### 7.7.4.13 SetLink()

```
void GUIComponent::Card::SetLink (
    std::function< void(States::ID) > link )
```

#### 7.7.4.14 SetPosition() [1/2]

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

##### Parameters

x	The x coordinate of the position.
y	The y coordinate of the position.

#### 7.7.4.15 SetPosition() [2/2]

```
void GUI::Component::SetPosition (
    Vector2 position )  [inherited]
```

Set the position of the component.

This function sets the position of the component.

##### Parameters

<i>position</i>	The position of the component.
-----------------	--------------------------------

#### 7.7.4.16 SetStateID()

```
void GUIComponent::Card::SetStateID (
    States::ID stateID )
```

#### 7.7.4.17 SetText()

```
void GUIComponent::Card::SetText (
    std::string text )
```

#### 7.7.4.18 SetVisible()

```
void GUI::Component::SetVisible (
    bool visible )  [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

##### Parameters

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

#### 7.7.4.19 ToggleVisible()

```
void GUI::Component::ToggleVisible () [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.7.4.20 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.7.4.21 UpdateMouseCursorWhenHover() [2/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

### 7.7.5 Member Data Documentation

#### 7.7.5.1 fonts

```
FontHolder* GUIComponent::Card::fonts [private]
```

### 7.7.5.2 hoverBounds

```
std::map< std::string, Rectangle > GUIComponent::Card::hoverBounds [private]
```

### 7.7.5.3 isHover

```
bool GUIComponent::Card::isHover [private]
```

### 7.7.5.4 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

### 7.7.5.5 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

### 7.7.5.6 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

### 7.7.5.7 stateID

```
States::ID GUIComponent::Card::stateID [private]
```

### 7.7.5.8 thumbnail

```
Texture GUIComponent::Card::thumbnail [private]
```

### 7.7.5.9 title

```
std::string GUIComponent::Card::title [private]
```

### 7.7.5.10 toLink

```
std::function< void(States::ID) > GUIComponent::Card::toLink [private]
```

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

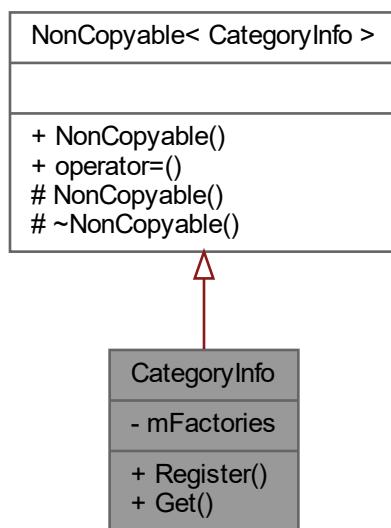
- src/Components/Common/[Card.hpp](#)
- src/Components/Common/[Card.cpp](#)

## 7.8 CategoryInfo Class Reference

The category info class that is used to store information about the categories.

```
#include <CategoryInfo.hpp>
```

Inheritance diagram for CategoryInfo:



## Classes

- struct [Info](#)

*The info struct that is used to store information about the categories.*

## Public Member Functions

- void [Register \(Category::ID, Info info\)](#)  
*This function is used to register a category.*
- const [Info & Get \(Category::ID id\) const](#)  
*This function is used to retrieve the information about a category.*

## Private Attributes

- std::map< [Category::ID](#), [Info](#) > [mFactories](#)  
*The factory map that is used to store the factories.*

### 7.8.1 Detailed Description

The category info class that is used to store information about the categories.

The category info class is used to store information about the categories. Also, this class is using the flyweight pattern to store the information about the categories.

#### See also

[Category::ID](#)  
[DataStructures::ID](#)  
[NonCopyable](#)  
[Info](#)

### 7.8.2 Member Function Documentation

#### 7.8.2.1 Get()

```
const CategoryInfo::Info & CategoryInfo::Get ( Category::ID id ) const
```

This function is used to retrieve the information about a category.

This function is used to retrieve the information about a category.

#### See also

[Category::ID](#)  
[Info](#)

**Parameters**

<i>id</i>	The category ID.
-----------	------------------

**Returns**

The information about the category.

**7.8.2.2 Register()**

```
void CategoryInfo::Register (
    Category::ID id,
    Info info )
```

This function is used to register a category.

This function is used to register a category.

**See also**

[Category::ID](#)

[Info](#)

**Parameters**

<i>id</i>	The category ID.
<i>info</i>	The information about the category.

**7.8.3 Member Data Documentation****7.8.3.1 mFactories**

```
std::map< Category::ID, Info > CategoryInfo::mFactories [private]
```

The factory map that is used to store the factories.

The factory map is used to store the factories. The factories are used to create the data structures.

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

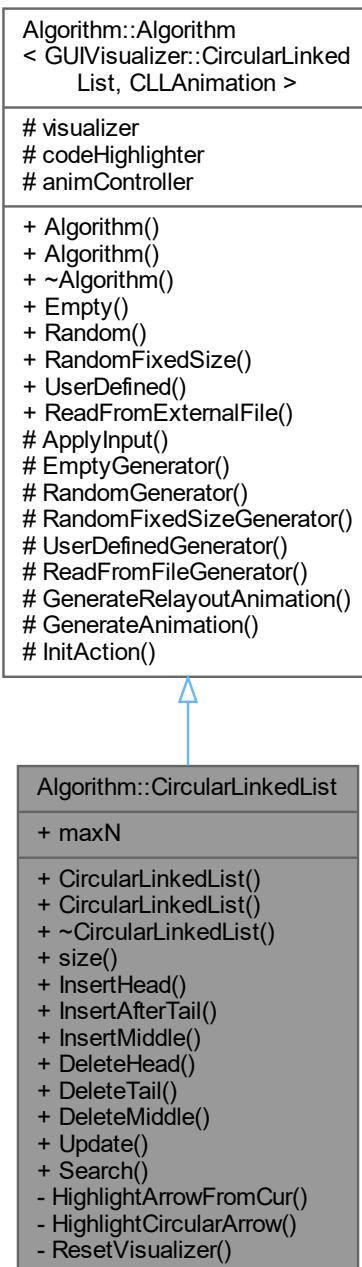
- src/Identifiers/[CategoryInfo.hpp](#)
- src/Identifiers/[CategoryInfo.cpp](#)

## 7.9 Algorithm::CircularLinkedList Class Reference

The algorithm class that is used to generate step-by-step instructions for the visualization of the circular linked list.

```
#include <CircularLinkedList.hpp>
```

Inheritance diagram for Algorithm::CircularLinkedList:



## Public Member Functions

- `CircularLinkedList ()`  
*The constructor of the circular linked list algorithm.*
- `CircularLinkedList (GUIComponent::CodeHighlighter::Ptr codeHighlighter, CLLAnimationController::Ptr animController, FontHolder *fonts)`  
*The constructor of the circular linked list algorithm.*
- `~CircularLinkedList ()`  
*The destructor of the circular linked list algorithm.*
- `std::size_t size () const`  
*Return the size of the circular linked list.*
- `void InsertHead (int value)`  
*Insert a new element before the head of the circular linked list.*
- `void InsertAfterTail (int value)`  
*Insert a new element after the tail of the circular linked list.*
- `void InsertMiddle (int index, int value)`  
*Insert a new element at the middle of the circular linked list.*
- `void DeleteHead ()`  
*Delete the head of the circular linked list.*
- `void DeleteTail ()`  
*Delete the tail of the circular linked list.*
- `void DeleteMiddle (int index)`  
*Delete the middle of the circular linked list.*
- `void Update (int index, int value)`  
*Update the value of the head of the circular linked list.*
- `void Search (int value)`  
*Search for an element in the circular linked list.*
- `virtual void Empty ()`  
*Initialize an empty data structure, and generate animation.*
- `virtual void Random ()`  
*Initialize a data structure with random values input, and then generate animation.*
- `virtual void RandomFixedSize (int N)`  
*Initialize a fixed size data structure with random values input, and then generate animation.*
- `virtual void UserDefined (std::string input)`  
*Initialize a data structure with input from user, and then generate animation.*
- `virtual void ReadFromExternalFile (std::string path)`  
*Initialize a data structure with input from external file (which is used to store the input), and then generate animation.*

## Static Public Attributes

- `static constexpr int maxN = 16`  
*The maximum number of elements that the circular linked list can hold.*

## Protected Member Functions

- virtual void [ApplyInput](#) (std::vector< int > input, std::size\_t nMaxSize=10)
 

*Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an std::vector< int >) to the data structure, used by other initialization functions.*
- std::vector< int > [EmptyGenerator](#) ()
 

*Generate an empty input, this is used to generate the input for [Empty\(\)](#)*
- std::vector< int > [RandomGenerator](#) ()
 

*Generate a random input, this is used to generate the input for [Random\(\)](#)*
- std::vector< int > [RandomFixedSizeGenerator](#) (int nSize)
 

*Generate a random input with fixed size, this is used to generate the input for [RandomFixedSize\(\)](#)*
- std::vector< int > [UserDefinedGenerator](#) (std::string input)
 

*Generate an input from user, this is used to generate the input for [UserDefined\(\)](#)*
- std::vector< int > [ReadFromFileGenerator](#) (std::string inputFile)
 

*Parse the input from external file, this is used to generate the input for [ReadFromExternalFile\(\)](#)*
- virtual void [GenerateRelayoutAnimation](#) (Vector2 newPosition)
 

*Generate the relayout animation (which reposition the data structure to a new position on the screen)*
- virtual [CLLAnimation GenerateAnimation](#) (float duration, int highlightLine, std::string actionDescription)
 

*Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)*
- virtual void [InitAction](#) (std::vector< std::string > code)
 

*Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.*

## Protected Attributes

- [GUIVisualizer::CircularLinkedList visualizer](#)

*Visualizer for the algorithm (which is used to draw animation generated by the algorithm)*
- [GUICodeHighlighter::Ptr codeHighlighter](#)

*Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)*
- [Animation::AnimationController< CLLAnimation >::Ptr animController](#)

*Animation controller for the algorithm (which is used to control the animation generated by the algorithm)*

## Private Member Functions

- std::function< [GUIVisualizer::CircularLinkedList\(GUIVisualizer::CircularLinkedList, float, Vector2\) >](#) [HighlightArrowFromCur](#) (int index, bool drawVisualizer=true, bool reverse=false)
 

*Generate a highlight arrow from the node at the given index to the next node.*
- std::function< [GUIVisualizer::CircularLinkedList\(GUIVisualizer::CircularLinkedList, float, Vector2\) >](#) [HighlightCircularArrow](#) (bool drawVisualizer=true, bool reverse=false)
 

*Generate a highlight arrow from the tail of the circular linked list to the head of the circular linked list.*
- void [ResetVisualizer](#) ()
 

*Reset the visualizer of the circular linked list algorithm.*

### 7.9.1 Detailed Description

The algorithm class that is used to generate step-by-step instructions for the visualization of the circular linked list.

## 7.9.2 Constructor & Destructor Documentation

### 7.9.2.1 CircularLinkedList() [1/2]

```
Algorithm::CircularLinkedList::CircularLinkedList ( )
```

The constructor of the circular linked list algorithm.

### 7.9.2.2 CircularLinkedList() [2/2]

```
Algorithm::CircularLinkedList::CircularLinkedList (
    GUIComponent::CodeHighlighter::Ptr codeHighlighter,
    CLLAnimationController::Ptr animController,
    FontHolder * fonts )
```

The constructor of the circular linked list algorithm.

#### Parameters

<i>codeHighlighter</i>	The code highlighter of the circular linked list algorithm.
<i>animController</i>	The animation controller of the circular linked list algorithm.
<i>fonts</i>	The fonts of the circular linked list algorithm.

### 7.9.2.3 ~CircularLinkedList()

```
Algorithm::CircularLinkedList::~CircularLinkedList ( )
```

The destructor of the circular linked list algorithm.

#### Note

This destructor is not virtual because this class is not designed to be inherited.

## 7.9.3 Member Function Documentation

### 7.9.3.1 ApplyInput()

```
void Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::ApplyInput (
    std::vector< int > input,
    std::size_t nMaxSize = 10 ) [protected], [virtual], [inherited]
```

Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an `std::vector< int >`) to the data structure, used by other initialization functions.

**Parameters**

<i>input</i>	the input (in std::vector< int >) to be applied to the data structure
<i>nMaxSize</i>	the maximum size of the data structure, if the input size is larger than nMaxSize, the input will be truncated to nMaxSize

**7.9.3.2 DeleteHead()**

```
void Algorithm::CircularLinkedList::DeleteHead ( )
```

Delete the head of the circular linked list.

**7.9.3.3 DeleteMiddle()**

```
void Algorithm::CircularLinkedList::DeleteMiddle (
    int index )
```

Delete the middle of the circular linked list.

**Parameters**

<i>index</i>	The index of the element to be deleted.
--------------	---

**Note**

This function is written as the visualization of deleting an element at the middle of the circular linked list is different from deleting an element at the head or tail of the circular linked list.

**7.9.3.4 DeleteTail()**

```
void Algorithm::CircularLinkedList::DeleteTail ( )
```

Delete the tail of the circular linked list.

**7.9.3.5 Empty()**

```
void Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::Empty [virtual],
[inherited]
```

Initialize an empty data structure, and generate animation.

### 7.9.3.6 EmptyGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::<
::EmptyGenerator [protected], [inherited]
```

Generate an empty input, this is used to generate the input for [Empty \(\)](#)

#### Returns

```
std::vector< int > the empty input
```

### 7.9.3.7 GenerateAnimation()

```
CLLAnimation Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::<
GenerateAnimation (
    float duration,
    int highlightLine,
    std::string actionDescription ) [protected], [virtual], [inherited]
```

Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)

#### Parameters

<i>duration</i>	the duration of the animation
<i>highlightLine</i>	the line to be highlighted in the code highlighter (if the line is -1, then no line will be highlighted)
<i>actionDescription</i>	the description of the action (which is used to display in the animation)

#### Returns

AnimationState the empty animation state (which currently only contains the metadata of the animation)

### 7.9.3.8 GenerateRelayoutAnimation()

```
void Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::Generate<
RelayoutAnimation (
    Vector2 newPosition ) [inline], [protected], [virtual], [inherited]
```

Generate the relayout animation (which reposition the data structure to a new position on the screen)

#### Parameters

<i>newPosition</i>	the new position of the data structure
--------------------	--

### 7.9.3.9 HighlightArrowFromCur()

```
std::function< GUIVisualizer::CircularLinkedList(GUIVisualizer::CircularLinkedList, float,
Vector2) > Algorithm::CircularLinkedList::HighlightArrowFromCur (
    int index,
    bool drawVisualizer = true,
    bool reverse = false ) [private]
```

Generate a highlight arrow from the node at the given index to the next node.

#### Parameters

<i>index</i>	The index of the node to be highlighted.
<i>drawVisualizer</i>	Whether to draw the visualizer, this is set to true by default.
<i>reverse</i>	Whether to reverse the animation (play the animation in reverse), this is set to false by default.

#### Returns

The animation of the highlight arrow.

### 7.9.3.10 HighlightCircularArrow()

```
std::function< GUIVisualizer::CircularLinkedList(GUIVisualizer::CircularLinkedList, float,
Vector2) > Algorithm::CircularLinkedList::HighlightCircularArrow (
    bool drawVisualizer = true,
    bool reverse = false ) [private]
```

Generate a highlight arrow from the tail of the circular linked list to the head of the circular linked list.

#### Parameters

<i>drawVisualizer</i>	Whether to draw the visualizer, this is set to true by default.
<i>reverse</i>	Whether to reverse the animation (play the animation in reverse), this is set to false by default.

#### Returns

The animation of the highlight arrow.

### 7.9.3.11 InitAction()

```
void Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::InitAction (
    std::vector< std::string > code ) [protected], [virtual], [inherited]
```

Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.

**Note**

This function is used to clear the animation controller and the code highlighter, and then initialize a new instruction

### 7.9.3.12 InsertAfterTail()

```
void Algorithm::CircularLinkedList::InsertAfterTail (
    int value )
```

Insert a new element after the tail of the circular linked list.

**Parameters**

<i>value</i>	The value of the new element.
--------------	-------------------------------

### 7.9.3.13 InsertHead()

```
void Algorithm::CircularLinkedList::InsertHead (
    int value )
```

Insert a new element before the head of the circular linked list.

**Parameters**

<i>value</i>	The value of the new element.
--------------	-------------------------------

### 7.9.3.14 InsertMiddle()

```
void Algorithm::CircularLinkedList::InsertMiddle (
    int index,
    int value )
```

Insert a new element at the middle of the circular linked list.

**Parameters**

<i>index</i>	The index of the new element.
<i>value</i>	The value of the new element.

**Note**

This function is written as the visualization of inserting a new element at the middle of the circular linked list is different from inserting a new element at the head or tail of the circular linked list.

### 7.9.3.15 Random()

```
void Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::Random [virtual],  
[inherited]
```

Initialize a data structure with random values input, and then generate animation.

### 7.9.3.16 RandomFixedSize()

```
void Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::RandomFixedSize  
(  
    int N ) [virtual], [inherited]
```

Initialize a fixed size data structure with random values input, and then generate animation.

**Parameters**

<i>N</i>	the size of the data structure
----------	--------------------------------

### 7.9.3.17 RandomFixedSizeGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >↔  
::RandomFixedSizeGenerator (  
    int nSize ) [protected], [inherited]
```

Generate a random input with fixed size, this is used to generate the input for `RandomFixedSize()`

**Parameters**

<i>nSize</i>	the size of the input
--------------	-----------------------

**Returns**

```
std::vector< int > the random input with fixed size
```

### 7.9.3.18 RandomGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::RandomGenerator [protected], [inherited]
```

Generate a random input, this is used to generate the input for [Random\(\)](#)

#### Returns

`std::vector< int >` the random input

### 7.9.3.19 ReadFromExternalFile()

```
void Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::ReadFrom<br>ExternalFile ( [virtual], [inherited]
```

Initialize a data structure with input from external file (which is used to store the input), and then generate animation.

#### Parameters

<code>path</code>	the path to the external file
-------------------	-------------------------------

### 7.9.3.20 ReadFromFileGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::ReadFromFileGenerator ( [protected], [inherited]
```

Parse the input from external file, this is used to generate the input for [ReadFromExternalFile\(\)](#)

#### Parameters

<code>inputFile</code>	the path to the external file
------------------------	-------------------------------

#### Returns

`std::vector< int >` the input from external file

### 7.9.3.21 ResetVisualizer()

```
void Algorithm::CircularLinkedList::ResetVisualizer ( ) [private]
```

Reset the visualizer of the circular linked list algorithm.

**Note**

This function is used to reset all of the effects that are only used to visualize the algorithm (i.e highlight the elements that are currently iterated).

### 7.9.3.22 Search()

```
void Algorithm::CircularLinkedList::Search (
    int value )
```

Search for an element in the circular linked list.

**Parameters**

<code>value</code>	The value of the element to be searched.
--------------------	--

### 7.9.3.23 size()

```
std::size_t Algorithm::CircularLinkedList::size ( ) const
```

Return the size of the circular linked list.

**Returns**

The size of the circular linked list.

### 7.9.3.24 Update()

```
void Algorithm::CircularLinkedList::Update (
    int index,
    int value )
```

Update the value of the head of the circular linked list.

**Parameters**

<code>value</code>	The new value of the head of the circular linked list.
--------------------	--

### 7.9.3.25 UserDefined()

```
void Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::UserDefined (
```

```
std::string input ) [virtual], [inherited]
```

Initialize a data structure with input from user, and then generate animation.

#### Parameters

<i>input</i>	the input from user
--------------	---------------------

### 7.9.3.26 UserDefinedGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::CircularLinkedList , CLLAnimation >::UserDefinedGenerator (
    std::string input ) [protected], [inherited]
```

Generate an input from user, this is used to generate the input for [UserDefined\(\)](#)

#### Parameters

<i>input</i>	the input from user
--------------	---------------------

#### Returns

std::vector< int > the input from user

## 7.9.4 Member Data Documentation

### 7.9.4.1 animController

```
Animation::AnimationController<CLLAnimation >::Ptr Algorithm::Algorithm< GUIVisualizer::CircularLinkedList ,
CLLAnimation >::animController [protected], [inherited]
```

[Animation](#) controller for the algorithm (which is used to control the animation generated by the algorithm)

#### Note

This [Algorithm](#) class is mainly use the animation controller to add animation for the algorithm

### 7.9.4.2 codeHighlighter

```
GUIComponent::CodeHighlighter::Ptr Algorithm::Algorithm< GUIVisualizer::CircularLinkedList ,
CLLAnimation >::codeHighlighter [protected], [inherited]
```

Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)

### 7.9.4.3 maxN

```
constexpr int Algorithm::CircularLinkedList::maxN = 16 [static], [constexpr]
```

The maximum number of elements that the circular linked list can hold.

### 7.9.4.4 visualizer

```
GUIVisualizer::CircularLinkedList Algorithm::Algorithm< GUIVisualizer::CircularLinkedList ,  
CLLAnimation >::visualizer [protected], [inherited]
```

Visualizer for the algorithm (which is used to draw animation generated by the algorithm)

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

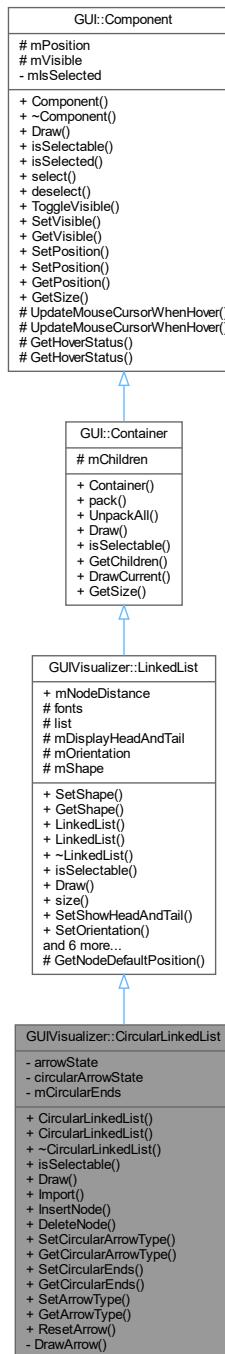
- src/Algorithms/LinkedList/CircularLinkedList.hpp
- src/Algorithms/LinkedList/CircularLinkedList.cpp

## 7.10 GUIVisualizer::CircularLinkedList Class Reference

The circular linked list visualization.

```
#include <CircularLinkedList.hpp>
```

Inheritance diagram for GUIVisualizer::CircularLinkedList:



## Public Types

- enum **ArrowType** {
 **Default** , **Hidden** , **Active** , **Skip** ,
 **ArrowTypeCount** }
- The type of the arrow.*
- enum **Orientation** { **Horizontal** , **Vertical** , **OrientationCount** }

- The orientation of the linked list.*
- `typedef std::shared_ptr< Container > Ptr`  
*The shared pointer to the container.*
- ## Public Member Functions
- `CircularLinkedList ()`  
*Construct a new Circular Linked List object.*
  - `CircularLinkedList (FontHolder *fonts)`  
*Construct a new Circular Linked List object.*
  - `~CircularLinkedList ()`  
*Destroy the Circular Linked List object.*
  - `bool IsSelectable () const`  
*Check if the container is selectable.*
  - `void Draw (Vector2 base=(Vector2){0, 0}, float t=1.0f, bool init=false)`  
*Draw the circular linked list visualization.*
  - `void Import (std::vector< int > nodes)`  
*Initialize the circular linked list visualization with the given nodes.*
  - `void InsertNode (std::size_t index, Node node, bool rePosition=true)`  
*Insert a node into the circular linked list visualization.*
  - `void DeleteNode (std::size_t index, bool rePosition=true)`  
*Delete a node from the circular linked list visualization.*
  - `void SetCircularArrowType (ArrowType type)`
  - `ArrowType GetCircularArrowType (std::size_t index)`
  - `void SetCircularEnds (std::size_t from, std::size_t to)`
  - `std::pair< std::size_t, std::size_t > GetCircularEnds ()`
  - `void SetArrowType (std::size_t index, ArrowType type)`
  - `ArrowType GetArrowType (std::size_t index)`
  - `void ResetArrow ()`
  - `void SetShape (Node::Shape shape)`  
*Set the shape of the node.*
  - `Node::Shape GetShape () const`  
*Get the shape of the node.*
  - `virtual void Draw (Vector2 base=(Vector2){0, 0})`  
*Draw the container.*
  - `virtual std::size_t Size () const`  
*Get the size of the linked list visualization.*
  - `virtual void SetShowHeadAndTail (bool show)`
  - `virtual void SetOrientation (Orientation orientation)`  
*Set the orientation of the linked list visualization.*
  - `virtual std::vector< Node > & GetList ()`
  - `virtual Node GenerateNode (int value)`
  - `virtual void Relayout ()`  
*relayout the linked list visualization.*
  - `void Pack (Component::Ptr component)`  
*Pack a component into the container.*
  - `void UnpackAll ()`  
*Unpack all components from the container.*
  - `Core::Deque< Component::Ptr > GetChildren ()`  
*Get the pack components of the container.*
  - `virtual void DrawCurrent (Vector2 base)`

- virtual Vector2 **GetSize** ()
 

*Get the size of the container.*
- bool **isSelected** () const
 

*Check if the component is selected.*
- virtual void **select** ()
 

*Select the component.*
- virtual void **deselect** ()
 

*Deselect the component.*
- virtual void **ToggleVisible** ()
 

*Toggle the visibility of the component.*
- virtual void **SetVisible** (bool visible)
 

*Set the visibility of the component.*
- virtual bool **GetVisible** ()
 

*Get the visibility of the component.*
- void **SetPosition** (float x, float y)
 

*Set the position of the component.*
- void **SetPosition** (Vector2 position)
 

*Set the position of the component.*
- Vector2 **GetPosition** ()
 

*Get the position of the component.*

## Static Public Attributes

- static constexpr float **mNodeDistance** = 20
 

*The distance between the [GUI](#) nodes.*

## Protected Member Functions

- Vector2 **GetNodeDefaultPosition** (std::size\_t index)
- virtual void **UpdateMouseCursorWhenHover** (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)
 

*Update the mouse cursor when hovering.*
- virtual void **UpdateMouseCursorWhenHover** (Rectangle bound, bool hover, bool noHover)
 

*Update the mouse cursor when hovering.*
- virtual bool **GetHoverStatus** (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)
 

*Get the hover status of the component.*
- virtual bool **GetHoverStatus** (Rectangle bound, bool hover, bool noHover)
 

*Get the hover status of the component.*

## Protected Attributes

- **FontHolder** \* fonts
- std::vector< **Node** > list
- bool **mDisplayHeadAndTail**
- Orientation **mOrientation** = Orientation::Horizontal
 

*The orientation of the linked list, by default it is horizontal.*
- **Node::Shape** **mShape** = **Node::Shape::Circle**

*The shape of the node, by default it is a circle.*
- **Core::Deque**< Component::Ptr > **mChildren**
- Vector2 **mPosition**

*The position of the component.*
- bool **mVisible**

*The visibility of the component.*

## Private Member Functions

- void [DrawArrow](#) (Vector2 base, float t)

## Private Attributes

- std::vector< [ArrowType](#) > arrowState
- [ArrowType](#) circularArrowState
- std::pair< std::size\_t, std::size\_t > mCircularEnds
- bool mIsSelected

*The selected status of the component.*

### 7.10.1 Detailed Description

The circular linked list visualization.

The circular linked list visualization is used to visualize the circular linked list.

### 7.10.2 Member Typedef Documentation

#### 7.10.2.1 Ptr

```
typedef std::shared_ptr< Container > GUI::Container::Ptr [inherited]
```

The shared pointer to the container.

The [GUI](#) container is commonly used by multiple components, so a shared pointer is used.

See also

[std::shared\\_ptr](#)

### 7.10.3 Member Enumeration Documentation

#### 7.10.3.1 ArrowType

```
enum GUIVisualizer::LinkedList::ArrowType [inherited]
```

The type of the arrow.

The type of the arrow is used to determine the color of the arrow.

**Enumerator**

Default	
Hidden	
Active	
Skip	
ArrowTypeCount	

**7.10.3.2 Orientation**

```
enum GUIVisualizer::LinkedList::Orientation [inherited]
```

The orientation of the linked list.

The orientation of the linked list is used to determine the orientation of the linked list.

**Enumerator**

Horizontal	
Vertical	
OrientationCount	

**7.10.4 Constructor & Destructor Documentation****7.10.4.1 CircularLinkedList() [1/2]**

```
GUIVisualizer::CircularLinkedList::CircularLinkedList ( )
```

Construct a new Circular Linked List object.

**7.10.4.2 CircularLinkedList() [2/2]**

```
GUIVisualizer::CircularLinkedList::CircularLinkedList (
    FontHolder * fonts )
```

Construct a new Circular Linked List object.

#### 7.10.4.3 ~CircularLinkedList()

```
GUIVisualizer::CircularLinkedList::~CircularLinkedList ( )
```

Destroy the Circular Linked List object.

### 7.10.5 Member Function Documentation

#### 7.10.5.1 DeleteNode()

```
void GUIVisualizer::CircularLinkedList::DeleteNode (   
    std::size_t index,  
    bool rePosition = true ) [virtual]
```

Delete a node from the circular linked list visualization.

##### Parameters

<i>index</i>	The index to delete the node.
<i>rePosition</i>	Whether to reposition the whole circular linked list after deletion.

##### See also

[LinkedList::DeleteNode](#)

Reimplemented from [GUIVisualizer::LinkedList](#).

#### 7.10.5.2 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

#### 7.10.5.3 Draw() [1/2]

```
void GUI::Container::Draw (   
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Draw the container.

This function draws the container.

**Parameters**

<i>base</i>	The base position of the container.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#).

**7.10.5.4 Draw() [2/2]**

```
void GUIVisualizer::CircularLinkedList::Draw (
    Vector2 base = (Vector2){0, 0},
    float t = 1.0f,
    bool init = false )  [virtual]
```

Draw the circular linked list visualization.

This function draws the circular linked list visualization.

**Parameters**

<i>base</i>	The base position of the circular linked list visualization.
<i>t</i>	The time delta.
<i>init</i>	True if the circular linked list visualization is initializing.

Implements [GUIVisualizer::LinkedList](#).

**7.10.5.5 DrawArrow()**

```
void GUIVisualizer::CircularLinkedList::DrawArrow (
    Vector2 base,
    float t )  [private]
```

**7.10.5.6 DrawCurrent()**

```
void GUI::Container::DrawCurrent (
    Vector2 base = (Vector2){0, 0} )  [virtual], [inherited]
```

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

### 7.10.5.7 GenerateNode()

```
GUIVisualizer::Node GUIVisualizer::CircularLinkedList::GenerateNode (
    int value ) [virtual], [inherited]
```

### 7.10.5.8 GetArrowType()

```
GUIVisualizer::LinkedList::ArrowType GUIVisualizer::CircularLinkedList::GetArrowType (
    std::size_t index )
```

### 7.10.5.9 GetChildren()

```
Core::Deque< GUI::Component::Ptr > GUI::Container::GetChildren () [inherited]
```

Get the pack components of the container.

This function returns the all of the pack components of the container.

#### Returns

The children of the container.

### 7.10.5.10 GetCircularArrowType()

```
GUIVisualizer::LinkedList::ArrowType GUIVisualizer::CircularLinkedList::GetCircularArrowType (
    std::size_t index )
```

### 7.10.5.11 GetCircularEnds()

```
std::pair< std::size_t, std::size_t > GUIVisualizer::CircularLinkedList::GetCircularEnds ()
```

### 7.10.5.12 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

**7.10.5.13 GetHoverStatus() [2/2]**

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.10.5.14 GetList()**

```
std::vector< GUIVisualizer::Node > & GUIVisualizer::LinkedList::GetList ( ) [virtual], [inherited]
```

**7.10.5.15 GetNodeDefaultPosition()**

```
Vector2 GUIVisualizer::LinkedList::GetNodeDefaultPosition (
    std::size_t index ) [protected], [inherited]
```

**7.10.5.16GetPosition()**

```
Vector2 GUI::Component::GetPosition ( ) [inherited]
```

Get the position of the component.

This function gets the position of the component.

**Returns**

The position of the component.

### 7.10.5.17 GetShape()

```
GUIVisualizer::Node::Shape GUIVisualizer::CircularLinkedList::GetShape ( ) const [inherited]
```

Get the shape of the node.

### 7.10.5.18 GetSize()

```
Vector2 GUI::Container::GetSize ( ) [virtual], [inherited]
```

Get the size of the container.

This function returns the size of the container.

#### Returns

The size of the container.

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#), and [GUIComponent::OptionInputField](#).

### 7.10.5.19 GetVisible()

```
bool GUI::Component::GetVisible ( ) [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

#### Returns

The visibility of the component.

### 7.10.5.20 Import()

```
void GUIVisualizer::CircularLinkedList::Import ( std::vector< int > nodes ) [virtual]
```

Initialize the circular linked list visualization with the given nodes.

#### Parameters

<code>nodes</code>	The nodes to initialize the circular linked list visualization with.
--------------------	--

See also

[LinkedList::Import](#)

Reimplemented from [GUIVisualizer::LinkedList](#).

#### 7.10.5.21 InsertNode()

```
void GUIVisualizer::CircularLinkedList::InsertNode (
    std::size_t index,
    GUIVisualizer::Node node,
    bool rePosition = true ) [virtual]
```

Insert a node into the circular linked list visualization.

Parameters

<i>index</i>	The index to insert the node.
<i>node</i>	The node to insert.
<i>rePosition</i>	Whether to reposition the whole circular linked list after insertion.

See also

[LinkedList::InsertNode](#)

Reimplemented from [GUIVisualizer::LinkedList](#).

#### 7.10.5.22 isSelectable()

```
bool GUIVisualizer::CircularLinkedList::isSelectable ( ) const [virtual]
```

Check if the container is selectable.

**Deprecated** This function is deprecated.

This function checks if the container is selectable.

Returns

True if the container is selectable.

Reimplemented from [GUI::Container](#).

### 7.10.5.23 isSelected()

```
bool GUI::Component::isSelected ( ) const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

#### Returns

True if the component is selected.

### 7.10.5.24 pack()

```
void GUI::Container::pack ( Component::Ptr component ) [inherited]
```

Pack a component into the container.

This function packs a component into the container.

#### Parameters

<code>component</code>	The component to pack.
------------------------	------------------------

### 7.10.5.25 Relayout()

```
void GUIVisualizer::LinkedList::Relayout ( ) [virtual], [inherited]
```

relayout the linked list visualization.

### 7.10.5.26 ResetArrow()

```
void GUIVisualizer::CircularLinkedList::ResetArrow ( )
```

### 7.10.5.27 **select()**

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

### 7.10.5.28 **SetArrowType()**

```
void GUIVisualizer::CircularLinkedList::SetArrowType (
    std::size_t index,
    ArrowType type )
```

### 7.10.5.29 **SetCircularArrowType()**

```
void GUIVisualizer::CircularLinkedList::SetCircularArrowType (
    ArrowType type )
```

### 7.10.5.30 **SetCircularEnds()**

```
void GUIVisualizer::CircularLinkedList::SetCircularEnds (
    std::size_t from,
    std::size_t to )
```

### 7.10.5.31 **SetOrientation()**

```
void GUIVisualizer::LinkedList::SetOrientation (
    Orientation orientation ) [virtual], [inherited]
```

Set the orientation of the linked list visualization.

#### Parameters

<i>orientation</i>	The orientation of the linked list visualization.
--------------------	---

### 7.10.5.32 SetPosition() [1/2]

```
void GUI::Component::SetPosition (
    float x,
    float y )  [inherited]
```

Set the position of the component.

This function sets the position of the component.

#### Parameters

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

### 7.10.5.33 SetPosition() [2/2]

```
void GUI::Component::SetPosition (
    Vector2 position )  [inherited]
```

Set the position of the component.

This function sets the position of the component.

#### Parameters

<i>position</i>	The position of the component.
-----------------	--------------------------------

### 7.10.5.34 SetShape()

```
void GUIVisualizer::LinkedList::SetShape (
    Node::Shape shape )  [inherited]
```

Set the shape of the node.

### 7.10.5.35 SetShowHeadAndTail()

```
void GUIVisualizer::LinkedList::SetShowHeadAndTail (
    bool show )  [virtual], [inherited]
```

### 7.10.5.36 SetVisible()

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

#### Parameters

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

### 7.10.5.37 size()

```
std::size_t GUIVisualizer::LinkedList::size () const [virtual], [inherited]
```

Get the size of the linked list visualization.

#### Returns

The size of the linked list visualization.

### 7.10.5.38 ToggleVisible()

```
void GUI::Component::ToggleVisible () [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

### 7.10.5.39 UnpackAll()

```
void GUI::Container::UnpackAll () [inherited]
```

Unpack all components from the container.

This function unpacks all components from the container.

#### See also

[pack](#)

### 7.10.5.40 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

**7.10.5.41 UpdateMouseCursorWhenHover() [2/2]**

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.10.6 Member Data Documentation****7.10.6.1 arrowState**

```
std::vector< ArrowType > GUIVisualizer::CircularLinkedList::arrowState [private]
```

**7.10.6.2 circularArrowState**

```
ArrowType GUIVisualizer::CircularLinkedList::circularArrowState [private]
```

**7.10.6.3 fonts**

```
FontHolder* GUIVisualizer::LinkedList::fonts [protected], [inherited]
```

#### 7.10.6.4 list

```
std::vector< Node > GUIVisualizer::LinkedList::list [protected], [inherited]
```

#### 7.10.6.5 mChildren

```
Core::Deque< Component::Ptr > GUI::Container::mChildren [protected], [inherited]
```

#### 7.10.6.6 mCircularEnds

```
std::pair< std::size_t, std::size_t > GUIVisualizer::CircularLinkedList::mCircularEnds [private]
```

#### 7.10.6.7 mDisplayHeadAndTail

```
bool GUIVisualizer::LinkedList::mDisplayHeadAndTail [protected], [inherited]
```

#### 7.10.6.8 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

#### 7.10.6.9 mNodeDistance

```
constexpr float GUIVisualizer::LinkedList::mNodeDistance = 20 [static], [constexpr], [inherited]
```

The distance between the **GUI** nodes.

### 7.10.6.10 mOrientation

```
Orientation GUIVisualizer::LinkedList::mOrientation = Orientation::Horizontal [protected],  
[inherited]
```

The orientation of the linked list, by default it is horizontal.

### 7.10.6.11 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

### 7.10.6.12 mShape

```
Node::Shape GUIVisualizer::LinkedList::mShape = Node::Shape::Circle [protected], [inherited]
```

The shape of the node, by default it is a circle.

### 7.10.6.13 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

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

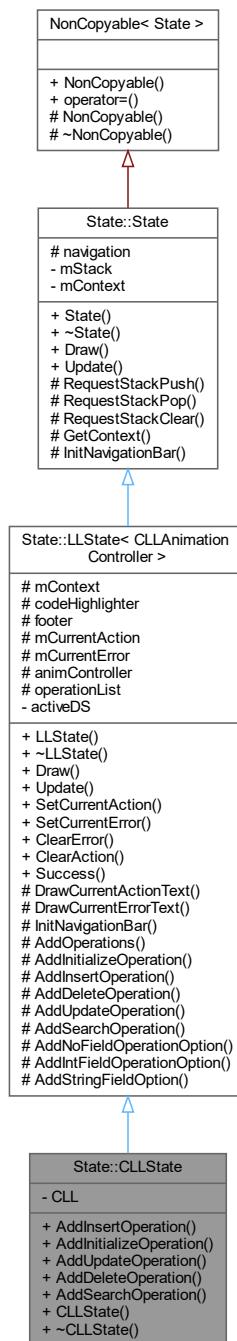
- src/Components/Visualization/[CircularLinkedList.hpp](#)
- src/Components/Visualization/[CircularLinkedList.cpp](#)

## 7.11 State::CLLState Class Reference

The state class that is used to represent the circular linked list state/scene of the application.

```
#include <CLLState.hpp>
```

Inheritance diagram for State::CLLState:



## Public Types

- `typedef std::unique_ptr< State > Ptr`

*A unique pointer to the state object.*

## Public Member Functions

- `void AddInsertOperation ()`  
*Add an insert operation to the circular linked list algorithm.*
- `void AddInitializeOperation ()`  
*Add an initialize operation to the circular linked list algorithm.*
- `void AddUpdateOperation ()`  
*Add an update operation to the circular linked list algorithm.*
- `void AddDeleteOperation ()`  
*Add a delete operation to the circular linked list algorithm.*
- `void AddSearchOperation ()`  
*Add a search operation to the circular linked list algorithm.*
- `CLLState (StateStack &stack, Context context)`  
*Construct a new `CLLState` object.*
- `~CLLState ()`  
*Destroy the `CLLState` object.*
- `virtual void Draw ()`  
*Draw the linked list state to the screen.*
- `virtual bool Update (float dt)`  
*Update the linked list state.*
- `virtual void SetCurrentAction (std::string action)`  
*Set the current action text.*
- `virtual void SetCurrentError (std::string error)`  
*Set the current error text.*
- `virtual void ClearError ()`  
*Clear the current error text.*
- `virtual void ClearAction ()`  
*Clear the current action text.*
- `virtual void Success ()`  
*Clear the current error and toggle the action list.*

## Protected Member Functions

- `virtual void DrawCurrentActionText ()`
- `virtual void DrawCurrentErrorText ()`
- `void InitNavigationBar ()`
- `virtual void AddOperations ()`  
*Add the operations to the operation list.*
- `virtual void AddNoFieldOperationOption (GUIComponent::OperationContainer::Ptr container, std::string title, std::function< void() > action)`
- `virtual void AddIntFieldOperationOption (GUIComponent::OperationContainer::Ptr container, std::string title, Core::Deque< IntegerInput > fields, std::function< void(std::map< std::string, std::string >) > action)`
- `virtual void AddStringFieldOption (GUIComponent::OperationContainer::Ptr container, std::string title, std::string label, std::function< void(std::map< std::string, std::string >) > action)`
- `void RequestStackPush (States::ID stateID)`

- Request the state stack to push a new state/scene.
- void [RequestStackPop \(\)](#)  
Request the state stack to pop the current state/scene.
- void [RequestStackClear \(\)](#)  
Request the state stack to clear all the states/scenes.
- [Context GetContext \(\) const](#)  
Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

## Protected Attributes

- [Context mContext](#)
- [GUIComponent::CodeHighlighter::Ptr codeHighlighter](#)
- [GUIComponent::Footer< CLLAnimationController > footer](#)
- std::string [mCurrentAction](#)
- std::string [mCurrentError](#)
- T::Ptr [animController](#)
- [GUIComponent::OperationList operationList](#)
- [GUIComponent::NavigationBar navigation](#)

## Private Attributes

- [Algorithm::CircularLinkedList CLL](#)  
*The algorithm of the circular linked list.*
- [DataStructures::ID activeDS](#)
- [StateStack \\* mStack](#)  
*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*

### 7.11.1 Detailed Description

The state class that is used to represent the circular linked list state/scene of the application.

### 7.11.2 Member Typedef Documentation

#### 7.11.2.1 Ptr

```
typedef std::unique_ptr< State > State::State::Ptr [inherited]
```

A unique pointer to the state object.

### 7.11.3 Constructor & Destructor Documentation

#### 7.11.3.1 CLLState()

```
State::CLLState::CLLState (
    StateStack & stack,
    Context context )
```

Construct a new [CLLState](#) object.

**Parameters**

<i>stack</i>	The state stack where the circular linked list state is pushed to.
<i>context</i>	The context of the application.

**7.11.3.2 ~CLLState()**

```
State::CLLState::~CLLState ( )
```

Destroy the [CLLState](#) object.

**7.11.4 Member Function Documentation****7.11.4.1 AddDeleteOperation()**

```
void State::CLLState::AddDeleteOperation ( ) [virtual]
```

Add a delete operation to the circular linked list algorithm.

Reimplemented from [State::LLState< CLLAnimationController >](#).

**7.11.4.2 AddInitializeOperation()**

```
void State::CLLState::AddInitializeOperation ( ) [virtual]
```

Add an initialize operation to the circular linked list algorithm.

Reimplemented from [State::LLState< CLLAnimationController >](#).

**7.11.4.3 AddInsertOperation()**

```
void State::CLLState::AddInsertOperation ( ) [virtual]
```

Add an insert operation to the circular linked list algorithm.

Reimplemented from [State::LLState< CLLAnimationController >](#).

#### 7.11.4.4 AddIntFieldOperationOption()

```
void State::LLState< CLLAnimationController >::AddIntFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    Core::Deque< IntegerInput > fields,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.11.4.5 AddNoFieldOperationOption()

```
void State::LLState< CLLAnimationController >::AddNoFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::function< void() > action ) [protected], [virtual], [inherited]
```

#### 7.11.4.6 AddOperations()

```
void State::LLState< CLLAnimationController >::AddOperations [protected], [virtual], [inherited]
```

Add the operations to the operation list.

##### Note

This function should not be overridden as it calls the other Add\*Operation functions.

#### 7.11.4.7 AddSearchOperation()

```
void State::CLLState::AddSearchOperation () [virtual]
```

Add a search operation to the circular linked list algorithm.

Reimplemented from [State::LLState< CLLAnimationController >](#).

#### 7.11.4.8 AddStringFieldOption()

```
void State::LLState< CLLAnimationController >::AddStringFieldOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::string label,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.11.4.9 AddUpdateOperation()

```
void State::CLLState::AddUpdateOperation ( ) [virtual]
```

Add an update operation to the circular linked list algorithm.

Reimplemented from [State::LLState< CLLAnimationController >](#).

#### 7.11.4.10 ClearAction()

```
void State::LLState< CLLAnimationController >::ClearAction [inline], [virtual], [inherited]
```

Clear the current action text.

#### 7.11.4.11 ClearError()

```
void State::LLState< CLLAnimationController >::ClearError [inline], [virtual], [inherited]
```

Clear the current error text.

#### 7.11.4.12 Draw()

```
void State::LLState< CLLAnimationController >::Draw [inline], [virtual], [inherited]
```

Draw the linked list state to the screen.

Implements [State::State](#).

#### 7.11.4.13 DrawCurrentActionText()

```
void State::LLState< CLLAnimationController >::DrawCurrentActionText [inline], [protected], [virtual], [inherited]
```

#### 7.11.4.14 DrawCurrentErrorText()

```
void State::LLState< CLLAnimationController >::DrawCurrentErrorText [inline], [protected], [virtual], [inherited]
```

#### 7.11.4.15 GetContext()

```
State::State::Context State::State::GetContext () const [protected], [inherited]
```

Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### Returns

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### See also

[Context](#)

#### 7.11.4.16 InitNavigationBar()

```
void State::LLState< CLLAnimationController >::InitNavigationBar [protected], [inherited]
```

#### 7.11.4.17 RequestStackClear()

```
void State::State::RequestStackClear () [protected], [inherited]
```

Request the state stack to clear all the states/scenes.

##### See also

[StateStack::ClearStates](#)

#### 7.11.4.18 RequestStackPop()

```
void State::State::RequestStackPop () [protected], [inherited]
```

Request the state stack to pop the current state/scene.

##### See also

[StateStack::PopState](#)

#### 7.11.4.19 RequestStackPush()

```
void State::State::RequestStackPush (
    States::ID stateID ) [protected], [inherited]
```

Request the state stack to push a new state/scene.

**Parameters**

<i>stateID</i>	The ID of the state/scene that will be pushed
----------------	---

**See also**[StateStack::PushState](#)**7.11.4.20 SetCurrentAction()**

```
void State::LLState< CLLAnimationController >::SetCurrentAction (
    std::string action ) [inline], [virtual], [inherited]
```

Set the current action text.

**Parameters**

<i>action</i>	The current action text.
---------------	--------------------------

**7.11.4.21 SetCurrentError()**

```
void State::LLState< CLLAnimationController >::SetCurrentError (
    std::string error ) [inline], [virtual], [inherited]
```

Set the current error text.

**Parameters**

<i>error</i>	The current error text.
--------------	-------------------------

**7.11.4.22 Success()**

```
void State::LLState< CLLAnimationController >::Success [inline], [virtual], [inherited]
```

Clear the current error and toggle the action list.

**7.11.4.23 Update()**

```
bool State::LLState< CLLAnimationController >::Update (
    float dt ) [virtual], [inherited]
```

Update the linked list state.

**Parameters**

<i>dt</i>	The delta time between two frames.
-----------	------------------------------------

**Returns**

true If the update was successful.  
false If the update was unsuccessful.

Implements [State::State](#).

## 7.11.5 Member Data Documentation

### 7.11.5.1 activeDS

`DataStructures::ID State::LLState< CLLAnimationController >::activeDS [private], [inherited]`

### 7.11.5.2 animController

`T::Ptr State::LLState< CLLAnimationController >::animController [protected], [inherited]`

### 7.11.5.3 CLL

`Algorithm::CircularLinkedList State::CLLState::CLL [private]`

The algorithm of the circular linked list.

### 7.11.5.4 codeHighlighter

`GUIComponent::CodeHighlighter::Ptr State::LLState< CLLAnimationController >::codeHighlighter [protected], [inherited]`

### 7.11.5.5 footer

`GUIComponent::Footer< CLLAnimationController > State::LLState< CLLAnimationController >::footer [protected], [inherited]`

### 7.11.5.6 mContext

```
Context State::LLState< CLLAnimationController >::mContext [protected], [inherited]
```

### 7.11.5.7 mCurrentAction

```
std::string State::LLState< CLLAnimationController >::mCurrentAction [protected], [inherited]
```

### 7.11.5.8 mCurrentError

```
std::string State::LLState< CLLAnimationController >::mCurrentError [protected], [inherited]
```

### 7.11.5.9 mStack

```
StateStack* State::State::mStack [private], [inherited]
```

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

### 7.11.5.10 navigation

```
GUIComponent::NavigationBar State::State::navigation [protected], [inherited]
```

### 7.11.5.11 operationList

```
GUIComponent::OperationList State::LLState< CLLAnimationController >::operationList [protected], [inherited]
```

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

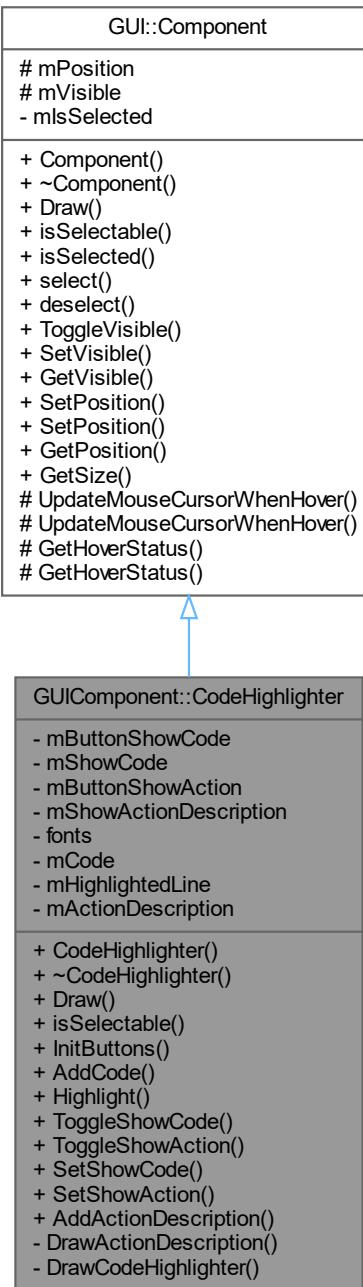
- src/States/LinkedList/[CLLState.hpp](#)
- src/States/LinkedList/[CLLState.cpp](#)

## 7.12 GUIComponent::CodeHighlighter Class Reference

The code highlighter class that is used to represent a code highlighter in the [GUI](#).

```
#include <CodeHighlighter.hpp>
```

Inheritance diagram for GUIComponent::CodeHighlighter:



### Public Types

- `typedef std::shared_ptr<CodeHighlighter> Ptr`

## Public Member Functions

- `CodeHighlighter (FontHolder *fonts)`
- `~CodeHighlighter ()`
- `void Draw (Vector2 base=(Vector2){0, 0})`  
*Draw the component.*
- `bool IsSelectable () const`  
*Check if the component is selectable.*
- `void InitButtons ()`
- `void AddCode (std::vector< std::string > code)`
- `void Highlight (int line)`
- `void ToggleShowCode ()`
- `void ToggleShowAction ()`
- `void SetShowCode (bool show)`
- `void SetShowAction (bool show)`
- `void AddActionDescription (std::string description)`
- `bool IsSelected () const`  
*Check if the component is selected.*
- `virtual void select ()`  
*Select the component.*
- `virtual void deselect ()`  
*Deselect the component.*
- `virtual void ToggleVisible ()`  
*Toggle the visibility of the component.*
- `virtual void SetVisible (bool visible)`  
*Set the visibility of the component.*
- `virtual bool GetVisible ()`  
*Get the visibility of the component.*
- `void SetPosition (float x, float y)`  
*Set the position of the component.*
- `void SetPosition (Vector2 position)`  
*Set the position of the component.*
- `Vector2 GetPosition ()`  
*Get the position of the component.*
- `virtual Vector2 GetSize ()`  
*Get the size of the component.*

## Protected Member Functions

- `virtual void UpdateMouseCursorWhenHover (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
*Update the mouse cursor when hovering.*
- `virtual void UpdateMouseCursorWhenHover (Rectangle bound, bool hover, bool noHover)`  
*Update the mouse cursor when hovering.*
- `virtual bool GetHoverStatus (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
*Get the hover status of the component.*
- `virtual bool GetHoverStatus (Rectangle bound, bool hover, bool noHover)`  
*Get the hover status of the component.*

## Protected Attributes

- `Vector2 mPosition`  
*The position of the component.*
- `bool mVisible`  
*The visibility of the component.*

## Private Member Functions

- `void DrawActionDescription (Vector2 base=(Vector2){0, 0})`
- `void DrawCodeHighlighter (Vector2 base=(Vector2){0, 0})`

## Private Attributes

- `GUIComponent::Button mButtonShowCode`
- `bool mShowCode`
- `GUIComponent::Button mButtonShowAction`
- `bool mShowActionDescription`
- `FontHolder * fonts`
- `std::vector< std::string > mCode`
- `int mHighlightedLine`
- `std::string mActionDescription`
- `bool mIsSelected`

*The selected status of the component.*

### 7.12.1 Detailed Description

The code highlighter class that is used to represent a code highlighter in the [GUI](#).

The code highlighter is used to show the code and highlight the current running line of code.

### 7.12.2 Member Typedef Documentation

#### 7.12.2.1 Ptr

```
typedef std::shared_ptr< CodeHighlighter > GUIComponent::CodeHighlighter::Ptr
```

### 7.12.3 Constructor & Destructor Documentation

### 7.12.3.1 CodeHighlighter()

```
GUIComponent::CodeHighlighter::CodeHighlighter (
    FontHolder * fonts )
```

### 7.12.3.2 ~CodeHighlighter()

```
GUIComponent::CodeHighlighter::~CodeHighlighter ( )
```

## 7.12.4 Member Function Documentation

### 7.12.4.1 AddActionDescription()

```
void GUIComponent::CodeHighlighter::AddActionDescription (
    std::string description )
```

### 7.12.4.2 AddCode()

```
void GUIComponent::CodeHighlighter::AddCode (
    std::vector< std::string > code )
```

### 7.12.4.3 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

### 7.12.4.4 Draw()

```
void GUIComponent::CodeHighlighter::Draw (
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Draw the component.

This function draws the component.

**Parameters**

<i>base</i>	The base position of the component.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

**7.12.4.5 DrawActionDescription()**

```
void GUIComponent::CodeHighlighter::DrawActionDescription (
    Vector2 base = (Vector2){0, 0} ) [private]
```

**7.12.4.6 DrawCodeHighlighter()**

```
void GUIComponent::CodeHighlighter::DrawCodeHighlighter (
    Vector2 base = (Vector2){0, 0} ) [private]
```

**7.12.4.7 GetHoverStatus() [1/2]**

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

**7.12.4.8 GetHoverStatus() [2/2]**

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.12.4.9 GetPosition()**

```
Vector2 GUI::Component::GetPosition () [inherited]
```

Get the position of the component.

This function gets the position of the component.

**Returns**

The position of the component.

**7.12.4.10 GetSize()**

```
Vector2 GUI::Component::GetSize () [virtual], [inherited]
```

Get the size of the component.

This function gets the size of the component.

**Returns**

The size of the component.

Reimplemented in [GUIComponent::Button](#), [GUIComponent::InputField](#), [GUIComponent::OperationList](#), [GUIComponent::OptionInputField](#) and [GUI::Container](#).

**7.12.4.11 GetVisible()**

```
bool GUI::Component::GetVisible () [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

**Returns**

The visibility of the component.

#### 7.12.4.12 **Highlight()**

```
void GUIComponent::CodeHighlighter::Highlight (
    int line )
```

#### 7.12.4.13 **InitButtons()**

```
void GUIComponent::CodeHighlighter::InitButtons ( )
```

#### 7.12.4.14 **isSelectable()**

```
bool GUIComponent::CodeHighlighter::isSelectable ( ) const [virtual]
```

Check if the component is selectable.

**Deprecated** This function is deprecated.

This function checks if the component is selectable.

##### Returns

True if the component is selectable.

Implements [GUI::Component](#).

#### 7.12.4.15 **isSelected()**

```
bool GUI::Component::isSelected ( ) const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.12.4.16 select()

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

#### 7.12.4.17 SetPosition() [1/2]

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

##### Parameters

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

#### 7.12.4.18 SetPosition() [2/2]

```
void GUI::Component::SetPosition (
    Vector2 position ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

##### Parameters

<i>position</i>	The position of the component.
-----------------	--------------------------------

#### 7.12.4.19 SetShowAction()

```
void GUIComponent::CodeHighlighter::SetShowAction (
    bool show )
```

#### 7.12.4.20 SetShowCode()

```
void GUIComponent::CodeHighlighter::SetShowCode (
    bool show )
```

#### 7.12.4.21 SetVisible()

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

##### Parameters

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

#### 7.12.4.22 ToggleShowAction()

```
void GUIComponent::CodeHighlighter::ToggleShowAction ( )
```

#### 7.12.4.23 ToggleShowCode()

```
void GUIComponent::CodeHighlighter::ToggleShowCode ( )
```

#### 7.12.4.24 ToggleVisible()

```
void GUI::Component::ToggleVisible ( ) [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.12.4.25 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

**7.12.4.26 UpdateMouseCursorWhenHover() [2/2]**

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.12.5 Member Data Documentation****7.12.5.1 fonts**

```
FontHolder* GUIComponent::CodeHighlighter::fonts [private]
```

**7.12.5.2 mActionDescription**

```
std::string GUIComponent::CodeHighlighter::mActionDescription [private]
```

**7.12.5.3 mButtonShowAction**

```
GUIComponent::Button GUIComponent::CodeHighlighter::mButtonShowAction [private]
```

#### 7.12.5.4 mButtonShowCode

```
GUIComponent::Button GUIComponent::CodeHighlighter::mButtonShowCode [private]
```

#### 7.12.5.5 mCode

```
std::vector< std::string > GUIComponent::CodeHighlighter::mCode [private]
```

#### 7.12.5.6 mHighlightedLine

```
int GUIComponent::CodeHighlighter::mHighlightedLine [private]
```

#### 7.12.5.7 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

#### 7.12.5.8 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

#### 7.12.5.9 mShowActionDescription

```
bool GUIComponent::CodeHighlighter::mShowActionDescription [private]
```

#### 7.12.5.10 mShowCode

```
bool GUIComponent::CodeHighlighter::mShowCode [private]
```

#### 7.12.5.11 mVisible

bool GUI::Component::mVisible [protected], [inherited]

The visibility of the component.

This variable stores the visibility of the component.

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

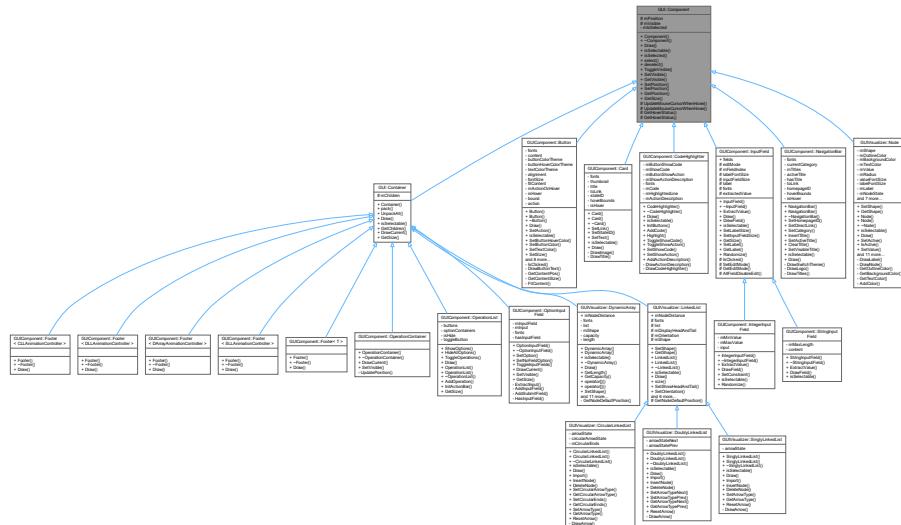
- src/Components/Common/CodeHighlighter.hpp
  - src/Components/Common/CodeHighlighter.cpp

## 7.13 GUI::Component Class Reference

The base component class that is used to represent a [GUI](#) component in the [GUI](#).

```
#include <Component.hpp>
```

## Inheritance diagram for GUI::Component:



## Public Types

- `typedef std::shared_ptr< Component > Ptr`  
*The shared pointer to the component.*

## Public Member Functions

- `Component ()`  
*Construct a new Component object.*
- `virtual ~Component ()`  
*Destroy the Component object.*
- `virtual void Draw (Vector2 base=(Vector2){0, 0})`  
*Draw the component.*
- `virtual bool IsSelectable () const =0`  
*Check if the component is selectable.*
- `bool IsSelected () const`  
*Check if the component is selected.*
- `virtual void Select ()`  
*Select the component.*
- `virtual void Deselect ()`  
*Deselect the component.*
- `virtual void ToggleVisible ()`  
*Toggle the visibility of the component.*
- `virtual void SetVisible (bool visible)`  
*Set the visibility of the component.*
- `virtual bool GetVisible ()`  
*Get the visibility of the component.*
- `void SetPosition (float x, float y)`  
*Set the position of the component.*
- `void SetPosition (Vector2 position)`  
*Set the position of the component.*
- `Vector2 GetPosition ()`  
*Get the position of the component.*
- `virtual Vector2 GetSize ()`  
*Get the size of the component.*

## Protected Member Functions

- `virtual void UpdateMouseCursorWhenHover (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
*Update the mouse cursor when hovering.*
- `virtual void UpdateMouseCursorWhenHover (Rectangle bound, bool hover, bool noHover)`  
*Update the mouse cursor when hovering.*
- `virtual bool GetHoverStatus (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
*Get the hover status of the component.*
- `virtual bool GetHoverStatus (Rectangle bound, bool hover, bool noHover)`  
*Get the hover status of the component.*

## Protected Attributes

- `Vector2 mPosition`  
*The position of the component.*
- `bool mVisible`  
*The visibility of the component.*

## Private Attributes

- bool `mIsSelected`

*The selected status of the component.*

### 7.13.1 Detailed Description

The base component class that is used to represent a [GUI](#) component in the [GUI](#).

### 7.13.2 Member Typedef Documentation

#### 7.13.2.1 Ptr

```
typedef std::shared_ptr< Component > GUI::Component::Ptr
```

The shared pointer to the component.

The [GUI](#) component is commonly used by multiple components, so a shared pointer is used.

See also

`std::shared_ptr`

### 7.13.3 Constructor & Destructor Documentation

#### 7.13.3.1 Component()

```
GUI::Component::Component ( )
```

Construct a new [Component](#) object.

#### 7.13.3.2 ~Component()

```
GUI::Component::~Component ( ) [virtual]
```

Destroy the [Component](#) object.

The destructor is virtual so that the derived classes can override it.

## 7.13.4 Member Function Documentation

### 7.13.4.1 deselect()

```
void GUI::Component::deselect ( ) [virtual]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

### 7.13.4.2 Draw()

```
void GUI::Component::Draw (
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Draw the component.

This function draws the component.

#### Parameters

<i>base</i>	The base position of the component.
-------------	-------------------------------------

Reimplemented in [GUIComponent::Button](#), [GUIComponent::Card](#), [GUIComponent::CodeHighlighter](#), [GUIComponent::InputField](#), [GUIComponent::NavigationBar](#), [GUIComponent::OperationList](#), and [GUI::Container](#).

### 7.13.4.3 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual]
```

Get the hover status of the component.

This function returns the hover status of the component.

#### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.13.4.4 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

#### 7.13.4.5 GetPosition()

```
Vector2 GUI::Component::GetPosition ()
```

Get the position of the component.

This function gets the position of the component.

##### Returns

The position of the component.

#### 7.13.4.6 GetSize()

```
Vector2 GUI::Component::GetSize () [virtual]
```

Get the size of the component.

This function gets the size of the component.

##### Returns

The size of the component.

Reimplemented in [GUIComponent::Button](#), [GUIComponent::InputField](#), [GUIComponent::OperationList](#), [GUIComponent::OptionInputField](#) and [GUI::Container](#).

#### 7.13.4.7 GetVisible()

```
bool GUI::Component::GetVisible ( ) [virtual]
```

Get the visibility of the component.

This function gets the visibility of the component.

##### Returns

The visibility of the component.

#### 7.13.4.8 isSelectable()

```
virtual bool GUI::Component::isSelectable ( ) const [pure virtual]
```

Check if the component is selectable.

**Deprecated** This function is deprecated.

This function checks if the component is selectable.

##### Returns

True if the component is selectable.

Implemented in [GUIComponent::Button](#), [GUIComponent::Card](#), [GUIComponent::CodeHighlighter](#), [GUIComponent::InputField](#), [GUIComponent::IntegerInputField](#), [GUIComponent::NavigationBar](#), [GUIComponent::StringInputField](#), [GUIVisualizer::CircularLinkedList](#), [GUIVisualizer::DoublyLinkedList](#), [GUIVisualizer::DynamicArray](#), [GUIVisualizer::LinkedList](#), [GUIVisualizer::Node](#), [GUIVisualizer::SinglyLinkedList](#), and [GUI::Container](#).

#### 7.13.4.9 isSelected()

```
bool GUI::Component::isSelected ( ) const
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.13.4.10 select()

```
void GUI::Component::select ( ) [virtual]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

#### 7.13.4.11 SetPosition() [1/2]

```
void GUI::Component::SetPosition (
    float x,
    float y )
```

Set the position of the component.

This function sets the position of the component.

##### Parameters

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

#### 7.13.4.12 SetPosition() [2/2]

```
void GUI::Component::SetPosition (
    Vector2 position )
```

Set the position of the component.

This function sets the position of the component.

##### Parameters

<i>position</i>	The position of the component.
-----------------	--------------------------------

#### 7.13.4.13 SetVisible()

```
void GUI::Component::SetVisible (
    bool visible ) [virtual]
```

Set the visibility of the component.

This function sets the visibility of the component.

**Parameters**

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

#### 7.13.4.14 ToggleVisible()

```
void GUI::Component::ToggleVisible ( ) [virtual]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.13.4.15 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.13.4.16 UpdateMouseCursorWhenHover() [2/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

## 7.13.5 Member Data Documentation

### 7.13.5.1 mIsSelected

```
bool GUI::Component::mIsSelected [private]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

### 7.13.5.2 mPosition

```
Vector2 GUI::Component::mPosition [protected]
```

The position of the component.

This variable stores the position of the component.

### 7.13.5.3 mVisible

```
bool GUI::Component::mVisible [protected]
```

The visibility of the component.

This variable stores the visibility of the component.

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

- src/[Component.hpp](#)
- src/[Component.cpp](#)

## 7.14 Core::List< T >::const\_iterator Class Reference

The list [const\\_iterator](#) class.

```
#include <List.hpp>
```

### Public Types

- using [iterator\\_category](#) = std::bidirectional\_iterator\_tag
- using [value\\_type](#) = T
- using [difference\\_type](#) = std::ptrdiff\_t
- using [pointer](#) = const T \*
- using [reference](#) = const T &

## Public Member Functions

- `const_iterator ()`  
`Construct a new const_iterator object.`
- `const_iterator (Node< value_type > *const &p)`  
`Construct a new const_iterator object.`
- `const_iterator (const iterator &other)`  
`Construct a new const_iterator object.`
- `const_iterator (const const_iterator &other)`  
`Construct a new const_iterator object.`
- `reference operator* () const`  
`Returns a reference to the object pointed to by the iterator.`
- `pointer operator-> () const`  
`Returns a pointer to the object pointed to by the iterator.`
- `const_iterator & operator++ ()`  
`Prefix increment.`
- `const_iterator operator++ (int)`  
`Postfix increment.`
- `const_iterator & operator-- ()`  
`Prefix decrement.`
- `const_iterator operator-- (int)`  
`Postfix decrement.`
- `const_iterator operator+ (const int &step)`  
`Returns an iterator after increased step steps.`
- `const_iterator operator- (const int &step)`  
`Returns an iterator after decreased step steps.`
- `const_iterator & operator= (Node< value_type > *const &p)`  
`Checks if the content of the iterator is equal to other.`
- `const_iterator & operator= (const iterator &other)`  
`Checks if two iterators are equal.`
- `bool operator== (const const_iterator &it) const`  
`Checks if two iterators are equal.`
- `bool operator!= (const const_iterator &it) const`  
`Checks if two iterators are not equal.`
- `void swap (const_iterator &other)`  
`Swaps the contents of the iterator with those of other.`

## Private Attributes

- `const Node< T > * ptr`

## Friends

- `class List`

### 7.14.1 Detailed Description

```
template<typename T>
class Core::List< T >::const_iterator
```

The list `const_iterator` class.

**Template Parameters**

<i>T</i>	The type of the elements
----------	--------------------------

## 7.14.2 Member Typedef Documentation

### 7.14.2.1 difference\_type

```
template<typename T >
using Core::List< T >::const_iterator::difference_type = std::ptrdiff_t
```

### 7.14.2.2 iterator\_category

```
template<typename T >
using Core::List< T >::const_iterator::iterator_category = std::bidirectional_iterator_tag
```

### 7.14.2.3 pointer

```
template<typename T >
using Core::List< T >::const_iterator::pointer = const T*
```

### 7.14.2.4 reference

```
template<typename T >
using Core::List< T >::const_iterator::reference = const T&
```

### 7.14.2.5 value\_type

```
template<typename T >
using Core::List< T >::const_iterator::value_type = T
```

## 7.14.3 Constructor & Destructor Documentation

#### 7.14.3.1 const\_iterator() [1/4]

```
template<typename T >
Core::List< T >::const_iterator::const_iterator ( )  [inline]
```

Construct a new `const_iterator` object.

#### 7.14.3.2 const\_iterator() [2/4]

```
template<typename T >
Core::List< T >::const_iterator::const_iterator (
    Node< value_type > *const & p )  [inline]
```

Construct a new `const_iterator` object.

##### Parameters

<code>p</code>	The pointer to the node
----------------	-------------------------

#### 7.14.3.3 const\_iterator() [3/4]

```
template<typename T >
Core::List< T >::const_iterator::const_iterator (
    const iterator & other )  [inline]
```

Construct a new `const_iterator` object.

##### Parameters

<code>other</code>	The iterator to copy
--------------------	----------------------

#### 7.14.3.4 const\_iterator() [4/4]

```
template<typename T >
Core::List< T >::const_iterator::const_iterator (
    const const_iterator & other )  [inline]
```

Construct a new `const_iterator` object.

##### Parameters

<code>other</code>	The <code>const_iterator</code> to copy
--------------------	---

## 7.14.4 Member Function Documentation

### 7.14.4.1 operator"!=()

```
template<typename T >
bool Core::List< T >::const_iterator::operator!= (
    const const_iterator & it ) const [inline]
```

Checks if two iterators are not equal.

#### Parameters

<i>it</i>	The iterator to compare with
-----------	------------------------------

#### Returns

true if the iterators are not equal

### 7.14.4.2 operator\*()

```
template<typename T >
reference Core::List< T >::const_iterator::operator* ( ) const [inline]
```

Returns a reference to the object pointed to by the iterator.

#### Returns

reference to the object pointed to by the iterator

### 7.14.4.3 operator+()

```
template<typename T >
const_iterator Core::List< T >::const_iterator::operator+ (
    const int & step ) [inline]
```

Returns an iterator after increased step steps.

#### Parameters

<i>step</i>	The number of steps to increase
-------------	---------------------------------

**Returns**

the resulting iterator

**7.14.4.4 operator++() [1/2]**

```
template<typename T >
const_iterator & Core::List< T >::const_iterator::operator++ ( ) [inline]
```

Prefix increment.

**Returns**

reference to the incremented iterator

**7.14.4.5 operator++() [2/2]**

```
template<typename T >
const_iterator Core::List< T >::const_iterator::operator++ (
    int ) [inline]
```

Postfix increment.

**Returns**

copy of the iterator before increment

**7.14.4.6 operator-()**

```
template<typename T >
const_iterator Core::List< T >::const_iterator::operator- (
    const int & step ) [inline]
```

Returns an iterator after decreased step steps.

**Parameters**

<i>step</i>	The number of steps to decrease
-------------	---------------------------------

**Returns**

the resulting iterator

#### 7.14.4.7 operator--() [1/2]

```
template<typename T >
const_iterator & Core::List< T >::const_iterator::operator-- ( ) [inline]
```

Prefix decrement.

##### Returns

reference to the decremented iterator

#### 7.14.4.8 operator--() [2/2]

```
template<typename T >
const_iterator Core::List< T >::const_iterator::operator-- (
    int ) [inline]
```

Postfix decrement.

##### Returns

copy of the iterator before decrement

#### 7.14.4.9 operator->()

```
template<typename T >
pointer Core::List< T >::const_iterator::operator-> ( ) const [inline]
```

Returns a pointer to the object pointed to by the iterator.

##### Returns

pointer to the object pointed to by the iterator

#### 7.14.4.10 operator=( ) [1/2]

```
template<typename T >
const_iterator & Core::List< T >::const_iterator::operator= (
    const iterator & other ) [inline]
```

Checks if two iterators are equal.

**Parameters**

<i>it</i>	The iterator to compare with
-----------	------------------------------

**Returns**

true if the iterators are equal

**7.14.4.11 operator=() [2/2]**

```
template<typename T >
const_iterator & Core::List< T >::const_iterator::operator= (
    Node< value_type > *const & p ) [inline]
```

Checks if the content of the iterator is equal to other.

**Parameters**

<i>other</i>	The pointer to the node to compare with
--------------	---

**Returns**

true if the content of the iterator is equal to the content pointed by other

**7.14.4.12 operator==( )**

```
template<typename T >
bool Core::List< T >::const_iterator::operator== (
    const const_iterator & it ) const [inline]
```

Checks if two iterators are equal.

**Parameters**

<i>it</i>	The iterator to compare with
-----------	------------------------------

**Returns**

true if the iterators are equal

#### 7.14.4.13 swap()

```
template<typename T >
void Core::List< T >::const_iterator::swap (
    const_iterator & other ) [inline]
```

Swaps the contents of the iterator with those of other.

##### Parameters

<i>other</i>	iterator to exchange the contents with
--------------	--

### 7.14.5 Friends And Related Function Documentation

#### 7.14.5.1 List

```
template<typename T >
friend class List [friend]
```

### 7.14.6 Member Data Documentation

#### 7.14.6.1 ptr

```
template<typename T >
const Node< T >* Core::List< T >::const_iterator::ptr [private]
```

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

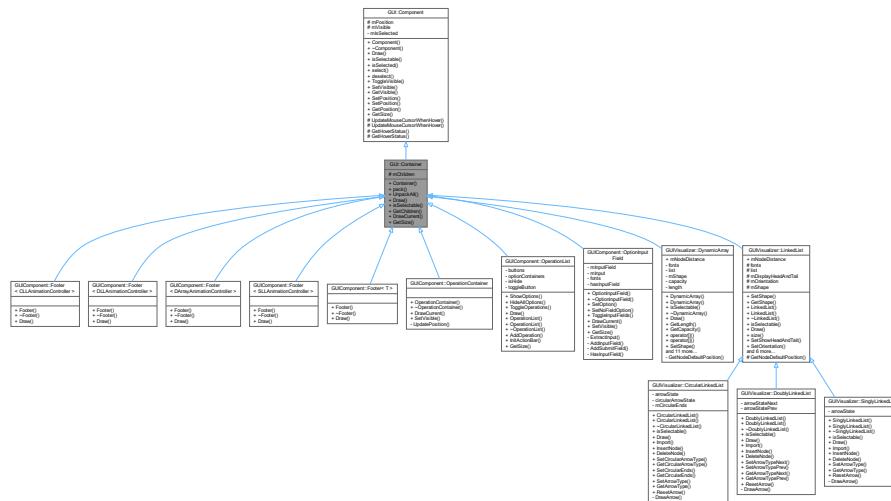
- src/Core/[List.hpp](#)

## 7.15 GUI::Container Class Reference

The base container class that is used to represent a [GUI](#) container in the [GUI](#).

```
#include <Container.hpp>
```

Inheritance diagram for GUI::Container:



## Public Types

- **typedef std::shared\_ptr< Container > Ptr**  
*The shared pointer to the container.*

## Public Member Functions

- **Container ()**  
*Construct a new Container object.*
- **void pack (Component::Ptr component)**  
*Pack a component into the container.*
- **void UnpackAll ()**  
*Unpack all components from the container.*
- **virtual void Draw (Vector2 base=(Vector2){0, 0})**  
*Draw the container.*
- **virtual bool isSelected () const**  
*Check if the container is selectable.*
- **Core::Deque< Component::Ptr > GetChildren ()**  
*Get the pack components of the container.*
- **virtual void DrawCurrent (Vector2 base)**
- **virtual Vector2 GetSize ()**  
*Get the size of the container.*
- **bool isSelected () const**  
*Check if the component is selected.*
- **virtual void select ()**  
*Select the component.*
- **virtual void deselect ()**  
*Deselect the component.*
- **virtual void ToggleVisible ()**  
*Toggle the visibility of the component.*
- **virtual void SetVisible (bool visible)**

- virtual bool [GetVisible \(\)](#)  
*Set the visibility of the component.*
- void [SetPosition \(float x, float y\)](#)  
*Get the visibility of the component.*
- void [SetPosition \(Vector2 position\)](#)  
*Set the position of the component.*
- Vector2 [GetPosition \(\)](#)  
*Get the position of the component.*

## Protected Member Functions

- virtual void [UpdateMouseCursorWhenHover \(std::map< std::string, Rectangle > bounds, bool hover, bool noHover\)](#)  
*Update the mouse cursor when hovering.*
- virtual void [UpdateMouseCursorWhenHover \(Rectangle bound, bool hover, bool noHover\)](#)  
*Update the mouse cursor when hovering.*
- virtual bool [GetHoverStatus \(std::map< std::string, Rectangle > bounds, bool hover, bool noHover\)](#)  
*Get the hover status of the component.*
- virtual bool [GetHoverStatus \(Rectangle bound, bool hover, bool noHover\)](#)  
*Get the hover status of the component.*

## Protected Attributes

- [Core::Deque< Component::Ptr > mChildren](#)
- Vector2 [mPosition](#)  
*The position of the component.*
- bool [mVisible](#)  
*The visibility of the component.*

## Private Attributes

- bool [mIsSelected](#)  
*The selected status of the component.*

### 7.15.1 Detailed Description

The base container class that is used to represent a [GUI](#) container in the [GUI](#).

### 7.15.2 Member Typedef Documentation

### 7.15.2.1 Ptr

```
typedef std::shared_ptr< Container > GUI::Container::Ptr
```

The shared pointer to the container.

The [GUI](#) container is commonly used by multiple components, so a shared pointer is used.

#### See also

```
std::shared_ptr
```

## 7.15.3 Constructor & Destructor Documentation

### 7.15.3.1 Container()

```
GUI::Container::Container ( )
```

Construct a new [Container](#) object.

## 7.15.4 Member Function Documentation

### 7.15.4.1 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

### 7.15.4.2 Draw()

```
void GUI::Container::Draw (   
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Draw the container.

This function draws the container.

**Parameters**

<i>base</i>	The base position of the container.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#).

**7.15.4.3 DrawCurrent()**

```
void GUI::Container::DrawCurrent (
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

**7.15.4.4 GetChildren()**

```
Core::Deque< GUI::Component::Ptr > GUI::Container::GetChildren ()
```

Get the pack components of the container.

This function returns the all of the pack components of the container.

**Returns**

The children of the container.

**7.15.4.5 GetHoverStatus() [1/2]**

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.15.4.6 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

#### 7.15.4.7 GetPosition()

```
Vector2 GUI::Component::GetPosition () [inherited]
```

Get the position of the component.

This function gets the position of the component.

##### Returns

The position of the component.

#### 7.15.4.8 GetSize()

```
Vector2 GUI::Container::GetSize () [virtual]
```

Get the size of the container.

This function returns the size of the container.

##### Returns

The size of the container.

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#), and [GUIComponent::OptionInputField](#).

#### 7.15.4.9 `GetVisible()`

```
bool GUI::Component::GetVisible ( ) [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

##### Returns

The visibility of the component.

#### 7.15.4.10 `isSelectable()`

```
bool GUI::Container::isSelectable ( ) const [virtual]
```

Check if the container is selectable.

**Deprecated** This function is deprecated.

This function checks if the container is selectable.

##### Returns

True if the container is selectable.

Implements [GUI::Component](#).

Reimplemented in [GUIVisualizer::CircularLinkedList](#), [GUIVisualizer::DoublyLinkedList](#), [GUIVisualizer::DynamicArray](#), [GUIVisualizer::LinkedList](#), and [GUIVisualizer::SinglyLinkedList](#).

#### 7.15.4.11 `isSelected()`

```
bool GUI::Component::isSelected ( ) const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.15.4.12 `pack()`

```
void GUI::Container::pack ( Component::Ptr component )
```

Pack a component into the container.

This function packs a component into the container.

**Parameters**

<i>component</i>	The component to pack.
------------------	------------------------

**7.15.4.13 select()**

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

**7.15.4.14 SetPosition() [1/2]**

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

**7.15.4.15 SetPosition() [2/2]**

```
void GUI::Component::SetPosition (
    Vector2 position ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>position</i>	The position of the component.
-----------------	--------------------------------

#### 7.15.4.16 SetVisible()

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

##### Parameters

<code>visible</code>	The visibility of the component.
----------------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

#### 7.15.4.17 ToggleVisible()

```
void GUI::Component::ToggleVisible () [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.15.4.18 UnpackAll()

```
void GUI::Container::UnpackAll ()
```

Unpack all components from the container.

This function unpacks all components from the container.

##### See also

[pack](#)

#### 7.15.4.19 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

**7.15.4.20 UpdateMouseCursorWhenHover() [2/2]**

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.15.5 Member Data Documentation****7.15.5.1 mChildren**

```
Core::Deque< Component::Ptr > GUI::Container::mChildren [protected]
```

**7.15.5.2 mIsSelected**

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

**7.15.5.3 mPosition**

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

#### 7.15.5.4 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

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

- src/[Container.hpp](#)
- src/[Container.cpp](#)

## 7.16 State::State::Context Struct Reference

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

```
#include <State.hpp>
```

### Public Member Functions

- [Context \(\)](#)
- [Context \(FontHolder \\*fonts, TextureHolder \\*textures, CategoryInfo \\*categories, DSInfo \\*dsInfo\)](#)

### Public Attributes

- [FontHolder \\* fonts](#)
- [TextureHolder \\* textures](#)
- [CategoryInfo \\* categories](#)
- [DSInfo \\* dsInfo](#)

#### 7.16.1 Detailed Description

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

#### 7.16.2 Constructor & Destructor Documentation

##### 7.16.2.1 Context() [1/2]

```
State::State::Context::Context ( )
```

### 7.16.2.2 Context() [2/2]

```
State::State::Context::Context (
    FontHolder * fonts,
    TextureHolder * textures,
    CategoryInfo * categories,
    DSInfo * dsInfo )
```

## 7.16.3 Member Data Documentation

### 7.16.3.1 categories

```
CategoryInfo* State::State::Context::categories
```

### 7.16.3.2 dsInfo

```
DSInfo* State::State::Context::dsInfo
```

### 7.16.3.3 fonts

```
FontHolder* State::State::Context::fonts
```

### 7.16.3.4 textures

```
TextureHolder* State::State::Context::textures
```

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

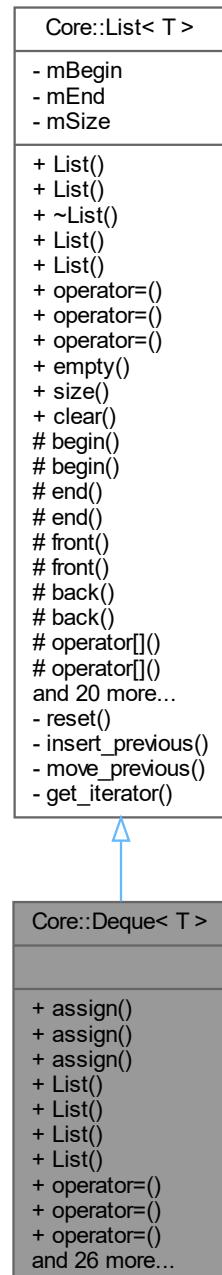
- src/[State.hpp](#)
- src/[State.cpp](#)

## 7.17 Core::Deque< T > Class Template Reference

The deque container.

```
#include <Deque.hpp>
```

Inheritance diagram for Core::Deque< T >:



## Public Member Functions

- void **assign** (std::size\_t count, const T &value)  
*Assigns new contents to the list, replacing its current content with count copies of value.*
- void **assign** (const const\_iterator &begin, const const\_iterator &end)  
*Assigns new contents to the list, replacing its current content.*
- void **assign** (std::initializer\_list< T > list)  
*Assigns new contents to the list, replacing its current content.*
- **List** ()  
*Default constructor.*
- **List** (std::initializer\_list< T > list)  
*Constructs the container with the contents of the initializer list.*
- **List** (const **List**< T > &list)  
*Copy constructor.*
- **List** (**List**< T > &&list)  
*Move constructor.*
- **List**< T > & **operator=** (std::initializer\_list< T > list)  
*Copy assign the container with the contents of the initializer list.*
- **List**< T > & **operator=** (const **List**< T > &list)  
*Copy assignment operator.*
- **List**< T > & **operator=** (**List**< T > &&list)  
*Move assignment operator.*
- void **swap** (**List**< T > &other)  
*Swaps the contents of the list with those of other.*
- void **clear** ()  
*Frees all elements in the container.*
- bool **empty** () const  
*Check whether the container is empty.*
- std::size\_t **size** () const  
*Returns the size of the container.*
- T & **back** ()  
*Returns the reference to the element at the back of the container.*
- const T & **back** () const  
*Returns the reference to the element at the back of the container.*
- T & **front** ()  
*Returns the reference to the element at the front of the container.*
- const T & **front** () const  
*Returns the reference to the element at the front of the container.*
- void **push\_back** (const T &value)  
*Pushes the element to the back of the container.*
- void **push\_front** (const T &value)  
*Pushes the element to the front of the container.*
- void **pop\_back** ()  
*Removes the element at the back of the container.*
- void **pop\_front** ()  
*Removes the element at the front of the container.*
- **iterator** **begin** ()
- **const\_iterator** **begin** () const
- **iterator** **end** ()
- **const\_iterator** **end** () const
- void **resize** (std::size\_t count)

- void **resize** (std::size\_t count, const T &value)
 

*Resizes the container to contain the given number of elements.*
- iterator **remove** (const iterator &it)
 

*Resizes the container to contain the given number of elements.*
- int **remove** (const T &value, const iterator &begin, const iterator &end)
 

*Removes the elements in the range [begin, end]*
- int **remove** (const T &value)
 

*Removes the elements that has the same value as the given one.*
- void **reverse** ()
 

*Reverses the order of the elements in the list.*
- std::size\_t **unique** ()
 

*Eliminates all except the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.*
- std::size\_t **unique** (std::function< bool(const T &, const T &) > predicate)
 

*Eliminates all except the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.*
- T & **at** (std::size\_t index)
 

*Returns the reference to the element at the given index.*
- const T & **at** (std::size\_t index) const
 

*Returns the reference to the element at the given index.*

## Protected Member Functions

- T & **operator[]** (std::size\_t index)
 

*Returns the reference to the element at the given index.*
- const T & **operator[]** (std::size\_t index) const
 

*Returns the reference to the element at the given index.*
- int **remove\_if** (std::function< bool(const T &) > predicate, const iterator &begin, const iterator &end)
- int **remove\_if** (std::function< bool(const T &) > predicate)
- void **swap** (List< T > &other)
 

*Swaps the contents of the list with those of other.*

## Private Types

- using List = Core::List< T >

## Private Member Functions

- void **reset** ()
- void **insert\_previous** (const iterator &it, const iterator &it\_prev)
 

*Insert an iterator before the specified iterator.*
- iterator **move\_previous** (const iterator &it, const iterator &first, const iterator &last)
- iterator **get\_iterator** (std::size\_t index)
 

*Returns an iterator to the element at index index*

## Private Attributes

- `iterator mBegin`  
*The head of the list.*
- `iterator mEnd`  
*The end of the list (one past the last element)*
- `std::size_t mSize`  
*The size of the list.*

### 7.17.1 Detailed Description

```
template<typename T>
class Core::Deque< T >
```

The deque container.

#### Template Parameters

<code>T</code>	the type of the elements
----------------	--------------------------

### 7.17.2 Member Typedef Documentation

#### 7.17.2.1 List

```
template<typename T >
using Core::Deque< T >::List = Core::List< T > [private]
```

### 7.17.3 Member Function Documentation

#### 7.17.3.1 assign() [1/3]

```
template<typename T >
void Core::List< T >::assign (
    const const_iterator & begin,
    const const_iterator & end ) [inline]
```

Assigns new contents to the list, replacing its current content.

#### Parameters

<code>begin</code>	The iterator to the first element to be assigned
<code>end</code>	The iterator to the last element to be assigned

### 7.17.3.2 assign() [2/3]

```
template<typename T >
void Core::List< T >::assign (
    std::initializer_list< T > list ) [inline]
```

Assigns new contents to the list, replacing its current content.

#### Parameters

<i>list</i>	The initializer list to be assigned
-------------	-------------------------------------

### 7.17.3.3 assign() [3/3]

```
template<typename T >
void Core::List< T >::assign (
    std::size_t count,
    const T & value ) [inline]
```

Assigns new contents to the list, replacing its current content with `count` copies of `value`

#### Parameters

<i>count</i>	The new size of the container
<i>value</i>	The value to be used to fill the container

### 7.17.3.4 at() [1/2]

```
template<typename T >
T & Core::List< T >::at (
    std::size_t index ) [inline]
```

Returns the reference to the element at the given index.

#### Parameters

<i>index</i>	The index of the element
--------------	--------------------------

#### Returns

T& The reference to the element at the given index

**Exceptions**

std::out\_of\_range: index out of range

**7.17.3.5 at() [2/2]**

```
template<typename T >
const T & Core::List< T >::at (
    std::size_t index ) const [inline]
```

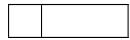
Returns the reference to the element at the given index.

**Parameters**

<i>index</i>	The index of the element
--------------	--------------------------

**Returns**

T& The reference to the element at the given index

**Exceptions**

std::out\_of\_range: index out of range

**7.17.3.6 back() [1/2]**

```
template<typename T >
T & Core::List< T >::back ( ) [inline]
```

Returns the reference to the element at the back of the container.

**Returns**

T& The reference to the element at the back of the container

**Exceptions**

undefined behavior: null pointer dereference

**7.17.3.7 back() [2/2]**

```
template<typename T >
```

```
const T & Core::List< T >::back ( ) const [inline]
```

Returns the reference to the element at the back of the container.

#### Returns

T& The reference to the element at the back of the container

#### Exceptions

--	--

undefined behavior: null pointer dereference

### 7.17.3.8 **begin()** [1/2]

```
template<typename T >
iterator Core::List< T >::begin ( ) [inline]
```

### 7.17.3.9 **begin()** [2/2]

```
template<typename T >
const_iterator Core::List< T >::begin ( ) const [inline]
```

### 7.17.3.10 **clear()**

```
template<typename T >
void Core::List< T >::clear ( ) [inline]
```

Frees all elements in the container.

### 7.17.3.11 **empty()**

```
template<typename T >
bool Core::List< T >::empty ( ) const [inline]
```

Check whether the container is empty.

#### Return values

<i>true</i>	The container is empty
<i>false</i>	The container is not empty

### 7.17.3.12 end() [1/2]

```
template<typename T >
iterator Core::List< T >::end ( ) [inline]
```

### 7.17.3.13 end() [2/2]

```
template<typename T >
const_iterator Core::List< T >::end ( ) const [inline]
```

### 7.17.3.14 front() [1/2]

```
template<typename T >
T & Core::List< T >::front ( ) [inline]
```

Returns the reference to the element at the front of the container.

#### Returns

T& The reference to the element at the front of the container

#### Exceptions



undefined behavior: null pointer dereference

### 7.17.3.15 front() [2/2]

```
template<typename T >
const T & Core::List< T >::front ( ) const [inline]
```

Returns the reference to the element at the front of the container.

#### Returns

T& The reference to the element at the front of the container

#### Exceptions



undefined behavior: null pointer dereference

#### 7.17.3.16 get\_iterator()

```
template<typename T >
iterator Core::List< T >::get_iterator (
    std::size_t index ) [inline], [private], [inherited]
```

Returns an iterator to the element at index `index`

##### Parameters

<code>index</code>	The index of the element
--------------------	--------------------------

##### Returns

An iterator to the element at index `index`

##### Exceptions

<code>std::out_of_range</code>	if `index` is out of range
--------------------------------	----------------------------

#### 7.17.3.17 insert\_previous()

```
template<typename T >
void Core::List< T >::insert_previous (
    const iterator & it,
    const iterator & it_prev ) [inline], [private], [inherited]
```

Insert an iterator before the specified iterator.

##### Parameters

<code>it</code>	The iterator to insert the iterator (prev) before
<code>it_prev</code>	The iterator to insert

#### 7.17.3.18 List() [1/4]

```
template<typename T >
Core::List< T >::List ( ) [inline]
```

Default constructor.

### 7.17.3.19 List() [2/4]

```
template<typename T >
Core::List< T >::List (
    const List< T > & list ) [inline]
```

Copy constructor.

#### Parameters

<i>rhs</i>	The other container
------------	---------------------

### 7.17.3.20 List() [3/4]

```
template<typename T >
Core::List< T >::List (
    List< T > && list ) [inline]
```

Move constructor.

#### Parameters

<i>rhs</i>	The other container
------------	---------------------

### 7.17.3.21 List() [4/4]

```
template<typename T >
Core::List< T >::List (
    std::initializer_list< T > list ) [inline]
```

Constructs the container with the contents of the initializer list.

#### Parameters

<i>init_list</i>	The initializer list
------------------	----------------------

### 7.17.3.22 move\_previous()

```
template<typename T >
iterator Core::List< T >::move_previous (
    const iterator & it,
    const iterator & first,
    const iterator & last ) [inline], [private], [inherited]
```

### 7.17.3.23 operator=( ) [1/3]

```
template<typename T >
List< T > & Core::List< T >::operator= (
    const List< T > & list ) [inline]
```

Copy assignment operator.

#### Parameters

<i>rhs</i>	The other container
------------	---------------------

### 7.17.3.24 operator=( ) [2/3]

```
template<typename T >
List< T > & Core::List< T >::operator= (
    List< T > && list ) [inline]
```

Move assignment operator.

#### Parameters

<i>rhs</i>	The other container
------------	---------------------

### 7.17.3.25 operator=( ) [3/3]

```
template<typename T >
List< T > & Core::List< T >::operator= (
    std::initializer_list< T > list ) [inline]
```

Copy assign the container with the contents of the initializer list.

#### Parameters

<i>init_list</i>	The initializer list
------------------	----------------------

### 7.17.3.26 operator[]( ) [1/2]

```
template<typename T >
T & Core::List< T >::operator[] (
    std::size_t index ) [inline], [protected], [inherited]
```

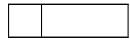
Returns the reference to the element at the given index.

**Parameters**

<i>index</i>	The index of the element
--------------	--------------------------

**Returns**

T& The reference to the element at the given index

**Exceptions**

std::out\_of\_range: index out of range

**7.17.3.27 operator[]( ) [2/2]**

```
template<typename T >
const T & Core::List< T >::operator[] ( std::size_t index ) const [inline], [protected], [inherited]
```

Returns the reference to the element at the given index.

**Parameters**

<i>index</i>	The index of the element
--------------	--------------------------

**Returns**

T& The reference to the element at the given index

**Exceptions**

std::out\_of\_range: index out of range

**7.17.3.28 pop\_back()**

```
template<typename T >
void Core::List< T >::pop_back ( ) [inline]
```

Removes the element at the back of the container.

### 7.17.3.29 `pop_front()`

```
template<typename T >
void Core::List< T >::pop_front ( ) [inline]
```

Removes the element at the front of the container.

### 7.17.3.30 `push_back()`

```
template<typename T >
void Core::List< T >::push_back (
    const T & value ) [inline]
```

Pushes the element to the back of the container.

#### Parameters

<code>elem</code>	The element to be pushed into the back
-------------------	--

### 7.17.3.31 `push_front()`

```
template<typename T >
void Core::List< T >::push_front (
    const T & value ) [inline]
```

Pushes the element to the front of the container.

#### Parameters

<code>elem</code>	The element to be pushed into the front
-------------------	---

### 7.17.3.32 `remove()` [1/3]

```
template<typename T >
iterator Core::List< T >::remove (
    const iterator & it ) [inline]
```

Removes the element iterator in the container.

#### Parameters

<code>it</code>	The iterator to the element to be removed
-----------------	---

**Returns**

iterator The iterator to the next element

**Exceptions**

<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------

std::out\_of\_range: iterator out of range

**7.17.3.33 remove() [2/3]**

```
template<typename T >
int Core::List< T >::remove (
    const T & value ) [inline]
```

Removes the elements that has the same value as the given one.

**Parameters**

<i>value</i>	The value of the elements to be removed
--------------	---

**Returns**

iterator The iterator to the next element

**7.17.3.34 remove() [3/3]**

```
template<typename T >
int Core::List< T >::remove (
    const T & value,
    const iterator & begin,
    const iterator & end ) [inline]
```

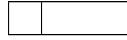
Removes the elements in the range [begin, end)

**Parameters**

<i>begin</i>	The iterator to the first element to be removed
<i>end</i>	The iterator to the last element to be removed

**Returns**

iterator The iterator to the next element

**Exceptions**

`std::out_of_range`: iterator out of range

**7.17.3.35 `remove_if()` [1/2]**

```
template<typename T >
int Core::List< T >::remove_if (
    std::function< bool(const T &) > predicate ) [inline], [protected], [inherited]
```

**7.17.3.36 `remove_if()` [2/2]**

```
template<typename T >
int Core::List< T >::remove_if (
    std::function< bool(const T &) > predicate,
    const iterator & begin,
    const iterator & end ) [inline], [protected], [inherited]
```

**7.17.3.37 `reset()`**

```
template<typename T >
void Core::List< T >::reset ( ) [inline], [private], [inherited]
```

**7.17.3.38 `resize()` [1/2]**

```
template<typename T >
void Core::List< T >::resize (
    std::size_t count ) [inline]
```

Resizes the container to contain the given number of elements.

**Parameters**

<code>count</code>	The new size of the container
--------------------	-------------------------------

**7.17.3.39 `resize()` [2/2]**

```
template<typename T >
```

```
void Core::List< T >::resize (
    std::size_t count,
    const T & value ) [inline]
```

Resizes the container to contain the given number of elements.

#### Parameters

<i>count</i>	The new size of the container
<i>value</i>	The value to be used to fill the container

### 7.17.3.40 reverse()

```
template<typename T >
void Core::List< T >::reverse ( ) [inline]
```

Reverses the order of the elements in the list.

### 7.17.3.41 size()

```
template<typename T >
std::size_t Core::List< T >::size ( ) const [inline]
```

Returns the size of the container.

#### Returns

The size of the container

### 7.17.3.42 swap() [1/2]

```
template<typename T >
void Core::List< T >::swap (
    List< T > & other ) [inline]
```

Swaps the contents of the list with those of other.

#### Parameters

<i>other</i>	list to exchange the contents with
--------------	------------------------------------

### 7.17.3.43 swap() [2/2]

```
template<typename T >
void Core::List< T >::swap (
    List< T > & other ) [inline], [protected], [inherited]
```

Swaps the contents of the list with those of other.

#### Parameters

<i>other</i>	list to exchange the contents with
--------------	------------------------------------

### 7.17.3.44 unique() [1/2]

```
template<typename T >
std::size_t Core::List< T >::unique ( ) [inline]
```

Eliminates all except the first element from every consecutive group of equivalent elements from the range [*first*, *last*) and returns a past-the-end iterator for the new logical end of the range.

#### Returns

the resulting size

### 7.17.3.45 unique() [2/2]

```
template<typename T >
std::size_t Core::List< T >::unique (
    std::function< bool(const T &, const T &) > predicate ) [inline]
```

Eliminates all except the first element from every consecutive group of equivalent elements from the range [*first*, *last*) and returns a past-the-end iterator for the new logical end of the range.

#### Parameters

<i>predicate</i>	Binary predicate that, taking two values of the same type than those contained in the list, returns true to remove the element passed as first argument from the container, and false otherwise.
------------------	--

#### Returns

the resulting size

## 7.17.4 Member Data Documentation

#### 7.17.4.1 mBegin

```
template<typename T >
iterator Core::List< T >::mBegin [private], [inherited]
```

The head of the list.

#### 7.17.4.2 mEnd

```
template<typename T >
iterator Core::List< T >::mEnd [private], [inherited]
```

The end of the list (one past the last element)

#### 7.17.4.3 mSize

```
template<typename T >
std::size_t Core::List< T >::mSize [private], [inherited]
```

The size of the list.

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

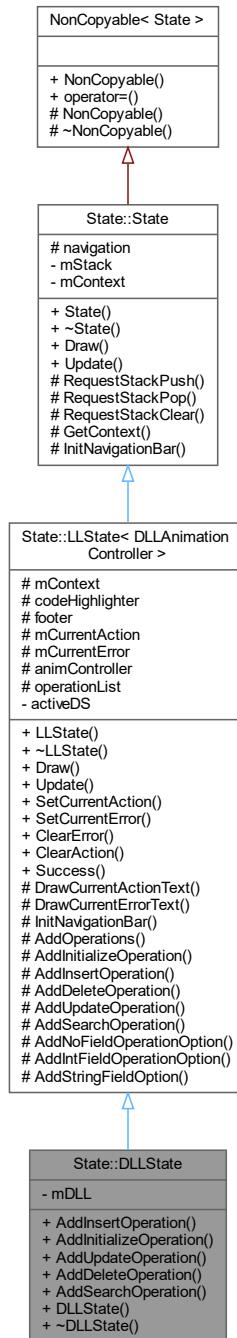
- src/Core/[Deque.hpp](#)

## 7.18 State::DLLState Class Reference

The state that is used to visualize the doubly linked list.

```
#include <DLLState.hpp>
```

Inheritance diagram for State::DLLState:



## Public Types

- `typedef std::unique_ptr< State > Ptr`

*A unique pointer to the state object.*

## Public Member Functions

- void [AddInsertOperation \(\)](#)  
*Add an insert operation to the doubly linked list algorithm.*
- void [AddInitializeOperation \(\)](#)  
*Add an initialize operation to the doubly linked list algorithm.*
- void [AddUpdateOperation \(\)](#)  
*Add an update operation to the doubly linked list algorithm.*
- void [AddDeleteOperation \(\)](#)  
*Add a delete operation to the doubly linked list algorithm.*
- void [AddSearchOperation \(\)](#)  
*Add a search operation to the doubly linked list algorithm.*
- [DLLState \(StateStack &stack, Context context\)](#)  
*Construct a new [DLLState](#) object.*
- [~DLLState \(\)](#)  
*Destroy the [DLLState](#) object.*
- virtual void [Draw \(\)](#)  
*Draw the linked list state to the screen.*
- virtual bool [Update \(float dt\)](#)  
*Update the linked list state.*
- virtual void [SetCurrentAction \(std::string action\)](#)  
*Set the current action text.*
- virtual void [SetCurrentError \(std::string error\)](#)  
*Set the current error text.*
- virtual void [ClearError \(\)](#)  
*Clear the current error text.*
- virtual void [ClearAction \(\)](#)  
*Clear the current action text.*
- virtual void [Success \(\)](#)  
*Clear the current error and toggle the action list.*

## Protected Member Functions

- virtual void [DrawCurrentActionText \(\)](#)
- virtual void [DrawCurrentErrorText \(\)](#)
- void [InitNavigationBar \(\)](#)
- virtual void [AddOperations \(\)](#)  
*Add the operations to the operation list.*
- virtual void [AddNoFieldOperationOption \(GUIComponent::OperationContainer::Ptr container, std::string title, std::function< void\(\) > action\)](#)
- virtual void [AddIntFieldOperationOption \(GUIComponent::OperationContainer::Ptr container, std::string title, Core::Deque< IntegerInput > fields, std::function< void\(std::map< std::string, std::string >\) > action\)](#)
- virtual void [AddStringFieldOption \(GUIComponent::OperationContainer::Ptr container, std::string title, std::string label, std::function< void\(std::map< std::string, std::string >\) > action\)](#)
- void [RequestStackPush \(States::ID stateID\)](#)  
*Request the state stack to push a new state/scene.*
- void [RequestStackPop \(\)](#)  
*Request the state stack to pop the current state/scene.*
- void [RequestStackClear \(\)](#)  
*Request the state stack to clear all the states/scenes.*
- [Context GetContext \(\) const](#)  
*Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).*

## Protected Attributes

- Context mContext
- GUIComponent::CodeHighlighter::Ptr codeHighlighter
- GUIComponent::Footer< DLLAnimationController > footer
- std::string mCurrentAction
- std::string mCurrentError
- T::Ptr animController
- GUIComponent::OperationList operationList
- GUIComponent::NavigationBar navigation

## Private Attributes

- Algorithm::DoublyLinkedList mDLL  
*The algorithm of the doubly linked list.*
- DataStructures::ID activeDS
- StateStack \* mStack  
*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*

### 7.18.1 Detailed Description

The state that is used to visualize the doubly linked list.

### 7.18.2 Member Typedef Documentation

#### 7.18.2.1 Ptr

```
typedef std::unique_ptr< State > State::State::Ptr [inherited]
```

A unique pointer to the state object.

### 7.18.3 Constructor & Destructor Documentation

#### 7.18.3.1 DLLState()

```
State::DLLState::DLLState (
    StateStack & stack,
    Context context )
```

Construct a new **DLLState** object.

**Parameters**

<i>stack</i>	The state stack where the doubly linked list state is pushed to.
<i>context</i>	The context of the application.

**7.18.3.2 ~DLLState()**

```
State::DLLState::~DLLState ( )
```

Destroy the [DLLState](#) object.

**7.18.4 Member Function Documentation****7.18.4.1 AddDeleteOperation()**

```
void State::DLLState::AddDeleteOperation ( ) [virtual]
```

Add a delete operation to the doubly linked list algorithm.

Reimplemented from [State::LLState< DLLAnimationController >](#).

**7.18.4.2 AddInitializeOperation()**

```
void State::DLLState::AddInitializeOperation ( ) [virtual]
```

Add an initialize operation to the doubly linked list algorithm.

Reimplemented from [State::LLState< DLLAnimationController >](#).

**7.18.4.3 AddInsertOperation()**

```
void State::DLLState::AddInsertOperation ( ) [virtual]
```

Add an insert operation to the doubly linked list algorithm.

Reimplemented from [State::LLState< DLLAnimationController >](#).

#### 7.18.4.4 AddIntFieldOperationOption()

```
void State::LLState< DLLAnimationController >::AddIntFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    Core::Deque< IntegerInput > fields,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.18.4.5 AddNoFieldOperationOption()

```
void State::LLState< DLLAnimationController >::AddNoFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::function< void() > action ) [protected], [virtual], [inherited]
```

#### 7.18.4.6 AddOperations()

```
void State::LLState< DLLAnimationController >::AddOperations [protected], [virtual], [inherited]
```

Add the operations to the operation list.

##### Note

This function should not be overridden as it calls the other Add\*Operation functions.

#### 7.18.4.7 AddSearchOperation()

```
void State::DLLState::AddSearchOperation () [virtual]
```

Add a search operation to the doubly linked list algorithm.

Reimplemented from [State::LLState< DLLAnimationController >](#).

#### 7.18.4.8 AddStringFieldOption()

```
void State::LLState< DLLAnimationController >::AddStringFieldOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::string label,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.18.4.9 AddUpdateOperation()

```
void State::DLLState::AddUpdateOperation ( ) [virtual]
```

Add an update operation to the doubly linked list algorithm.

Reimplemented from [State::LLState< DLLAnimationController >](#).

#### 7.18.4.10 ClearAction()

```
void State::LLState< DLLAnimationController >::ClearAction [inline], [virtual], [inherited]
```

Clear the current action text.

#### 7.18.4.11 ClearError()

```
void State::LLState< DLLAnimationController >::ClearError [inline], [virtual], [inherited]
```

Clear the current error text.

#### 7.18.4.12 Draw()

```
void State::LLState< DLLAnimationController >::Draw [inline], [virtual], [inherited]
```

Draw the linked list state to the screen.

Implements [State::State](#).

#### 7.18.4.13 DrawCurrentActionText()

```
void State::LLState< DLLAnimationController >::DrawCurrentActionText [inline], [protected],  
[virtual], [inherited]
```

#### 7.18.4.14 DrawCurrentErrorText()

```
void State::LLState< DLLAnimationController >::DrawCurrentErrorText [inline], [protected],  
[virtual], [inherited]
```

#### 7.18.4.15 GetContext()

```
State::State::Context State::State::GetContext () const [protected], [inherited]
```

Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### Returns

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### See also

[Context](#)

#### 7.18.4.16 InitNavigationBar()

```
void State::LLState< DLLAnimationController >::InitNavigationBar [protected], [inherited]
```

#### 7.18.4.17 RequestStackClear()

```
void State::State::RequestStackClear () [protected], [inherited]
```

Request the state stack to clear all the states/scenes.

##### See also

[StateStack::ClearStates](#)

#### 7.18.4.18 RequestStackPop()

```
void State::State::RequestStackPop () [protected], [inherited]
```

Request the state stack to pop the current state/scene.

##### See also

[StateStack::PopState](#)

#### 7.18.4.19 RequestStackPush()

```
void State::State::RequestStackPush (
    States::ID stateID ) [protected], [inherited]
```

Request the state stack to push a new state/scene.

**Parameters**

<i>stateID</i>	The ID of the state/scene that will be pushed
----------------	---

**See also**[StateStack::PushState](#)**7.18.4.20 SetCurrentAction()**

```
void State::LLState< DLLAnimationController >::SetCurrentAction (
    std::string action) [inline], [virtual], [inherited]
```

Set the current action text.

**Parameters**

<i>action</i>	The current action text.
---------------	--------------------------

**7.18.4.21 SetCurrentError()**

```
void State::LLState< DLLAnimationController >::SetCurrentError (
    std::string error) [inline], [virtual], [inherited]
```

Set the current error text.

**Parameters**

<i>error</i>	The current error text.
--------------	-------------------------

**7.18.4.22 Success()**

```
void State::LLState< DLLAnimationController >::Success [inline], [virtual], [inherited]
```

Clear the current error and toggle the action list.

**7.18.4.23 Update()**

```
bool State::LLState< DLLAnimationController >::Update (
    float dt) [virtual], [inherited]
```

Update the linked list state.

**Parameters**

<i>dt</i>	The delta time between two frames.
-----------	------------------------------------

**Returns**

true If the update was successful.  
false If the update was unsuccessful.

Implements [State::State](#).

## 7.18.5 Member Data Documentation

### 7.18.5.1 activeDS

```
DataStructures::ID State::LLState< DLLAnimationController >::activeDS [private], [inherited]
```

### 7.18.5.2 animController

```
T::Ptr State::LLState< DLLAnimationController >::animController [protected], [inherited]
```

### 7.18.5.3 codeHighlighter

```
GUIComponent::CodeHighlighter::Ptr State::LLState< DLLAnimationController >::codeHighlighter  
[protected], [inherited]
```

### 7.18.5.4 footer

```
GUIComponent::Footer< DLLAnimationController > State::LLState< DLLAnimationController >↔  
::footer [protected], [inherited]
```

### 7.18.5.5 mContext

```
Context State::LLState< DLLAnimationController >::mContext [protected], [inherited]
```

### 7.18.5.6 mCurrentAction

```
std::string State::LLState< DLLAnimationController >::mCurrentAction [protected], [inherited]
```

### 7.18.5.7 mCurrentError

```
std::string State::LLState< DLLAnimationController >::mCurrentError [protected], [inherited]
```

### 7.18.5.8 mDLL

```
Algorithm::DoublyLinkedList State::DLLState::mDLL [private]
```

The algorithm of the doubly linked list.

### 7.18.5.9 mStack

```
StateStack* State::State::mStack [private], [inherited]
```

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

### 7.18.5.10 navigation

```
GUIComponent::NavigationBar State::State::navigation [protected], [inherited]
```

### 7.18.5.11 operationList

```
GUIComponent::OperationList State::LLState< DLLAnimationController >::operationList [protected], [inherited]
```

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

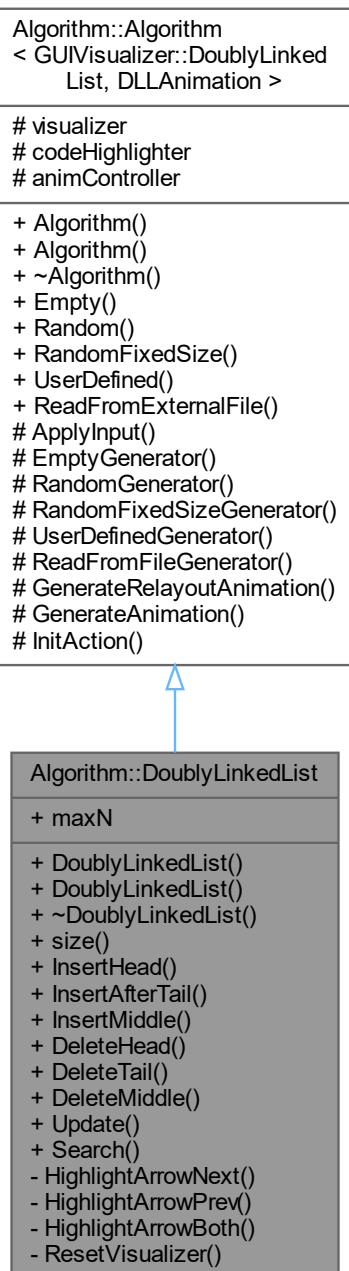
- src/States/LinkedList/[DLLState.hpp](#)
- src/States/LinkedList/[DLLState.cpp](#)

## 7.19 Algorithm::DoublyLinkedList Class Reference

The algorithm class that is used to generate step-by-step instructions for the visualization of the doubly linked list.

```
#include <DoublyLinkedList.hpp>
```

Inheritance diagram for Algorithm::DoublyLinkedList:



## Public Member Functions

- **DoublyLinkedList ()**  
*The constructor of the doubly linked list algorithm.*
- **DoublyLinkedList (GUIComponent::CodeHighlighter::Ptr codeHighlighter, DLLAnimationController::Ptr animController, FontHolder \*fonts)**  
*The constructor of the doubly linked list algorithm.*
- **~DoublyLinkedList ()**  
*The destructor of the doubly linked list algorithm.*
- **std::size\_t size () const**  
*Return the size of the doubly linked list.*
- **void InsertHead (int value)**  
*Insert a new element before the head of the doubly linked list.*
- **void InsertAfterTail (int value)**  
*Insert a new element after the tail of the doubly linked list.*
- **void InsertMiddle (int index, int value)**  
*Insert a new element at the middle of the doubly linked list.*
- **void DeleteHead ()**  
*Delete the head of the doubly linked list.*
- **void DeleteTail ()**  
*Delete the tail of the doubly linked list.*
- **void DeleteMiddle (int index)**  
*Delete an element at the middle of the doubly linked list.*
- **void Update (int index, int value)**  
*Update the value of an element at the head of the doubly linked list.*
- **void Search (int value)**  
*Search for an element in the doubly linked list.*
- **virtual void Empty ()**  
*Initialize an empty data structure, and generate animation.*
- **virtual void Random ()**  
*Initialize a data structure with random values input, and then generate animation.*
- **virtual void RandomFixedSize (int N)**  
*Initialize a fixed size data structure with random values input, and then generate animation.*
- **virtual void UserDefined (std::string input)**  
*Initialize a data structure with input from user, and then generate animation.*
- **virtual void ReadFromExternalFile (std::string path)**  
*Initialize a data structure with input from external file (which is used to store the input), and then generate animation.*

## Static Public Attributes

- **static constexpr int maxN = 16**  
*The maximum number of elements that the doubly linked list can hold.*

## Protected Member Functions

- virtual void [ApplyInput](#) (std::vector< int > input, std::size\_t nMaxSize=10)
 

*Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an std::vector< int >) to the data structure, used by other initialization functions.*
- std::vector< int > [EmptyGenerator](#) ()
 

*Generate an empty input, this is used to generate the input for [Empty\(\)](#)*
- std::vector< int > [RandomGenerator](#) ()
 

*Generate a random input, this is used to generate the input for [Random\(\)](#)*
- std::vector< int > [RandomFixedSizeGenerator](#) (int nSize)
 

*Generate a random input with fixed size, this is used to generate the input for [RandomFixedSize\(\)](#)*
- std::vector< int > [UserDefinedGenerator](#) (std::string input)
 

*Generate an input from user, this is used to generate the input for [UserDefined\(\)](#)*
- std::vector< int > [ReadFromFileGenerator](#) (std::string inputFile)
 

*Parse the input from external file, this is used to generate the input for [ReadFromExternalFile\(\)](#)*
- virtual void [GenerateRelayoutAnimation](#) (Vector2 newPosition)
 

*Generate the relayout animation (which reposition the data structure to a new position on the screen)*
- virtual [DLLAnimation GenerateAnimation](#) (float duration, int highlightLine, std::string actionDescription)
 

*Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)*
- virtual void [InitAction](#) (std::vector< std::string > code)
 

*Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.*

## Protected Attributes

- [GUIVisualizer::DoublyLinkedList](#) visualizer
 

*Visualizer for the algorithm (which is used to draw animation generated by the algorithm)*
- [GUIComponent::CodeHighlighter::Ptr](#) codeHighlighter
 

*Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)*
- [Animation::AnimationController< DLLAnimation >::Ptr](#) animController
 

*Animation controller for the algorithm (which is used to control the animation generated by the algorithm)*

## Private Member Functions

- std::function< [GUIVisualizer::DoublyLinkedList](#)([GUIVisualizer::DoublyLinkedList](#), float, Vector2) > [HighlightArrowNext](#) (int index, bool drawVisualizer=true, bool reverse=false)
 

*Generate a highlight arrow from the node at the given index to the next node of the doubly linked list.*
- std::function< [GUIVisualizer::DoublyLinkedList](#)([GUIVisualizer::DoublyLinkedList](#), float, Vector2) > [HighlightArrowPrev](#) (int index, bool drawVisualizer=true, bool reverse=false)
 

*Generate a highlight arrow from the next node at the given index to the node at the given index of the doubly linked list.*
- std::function< [GUIVisualizer::DoublyLinkedList](#)([GUIVisualizer::DoublyLinkedList](#), float, Vector2) > [HighlightArrowBoth](#) (int index, bool drawVisualizer=true, bool reverse=false)
 

*Generate a highlight arrow from the node at the given index to the previous node of the doubly linked list (in both direction).*
- void [ResetVisualizer](#) ()
 

*Reset the visualizer of the doubly linked list algorithm.*

### 7.19.1 Detailed Description

The algorithm class that is used to generate step-by-step instructions for the visualization of the doubly linked list.

## 7.19.2 Constructor & Destructor Documentation

### 7.19.2.1 DoublyLinkedList() [1/2]

```
Algorithm::DoublyLinkedList::DoublyLinkedList ( )
```

The constructor of the doubly linked list algorithm.

### 7.19.2.2 DoublyLinkedList() [2/2]

```
Algorithm::DoublyLinkedList::DoublyLinkedList (
    GUIComponent::CodeHighlighter::Ptr codeHighlighter,
    DLLAnimationController::Ptr animController,
    FontHolder * fonts )
```

The constructor of the doubly linked list algorithm.

#### Parameters

<i>codeHighlighter</i>	The code highlighter of the doubly linked list algorithm.
<i>animController</i>	The animation controller of the doubly linked list algorithm.
<i>fonts</i>	The fonts of the doubly linked list algorithm.

### 7.19.2.3 ~DoublyLinkedList()

```
Algorithm::DoublyLinkedList::~DoublyLinkedList ( )
```

The destructor of the doubly linked list algorithm.

#### Note

This destructor is not virtual because this class is not designed to be inherited.

## 7.19.3 Member Function Documentation

### 7.19.3.1 ApplyInput()

```
void Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::ApplyInput (
    std::vector< int > input,
    std::size_t nMaxSize = 10 ) [protected], [virtual], [inherited]
```

Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an `std::vector< int >`) to the data structure, used by other initialization functions.

**Parameters**

<i>input</i>	the input (in std::vector< int >) to be applied to the data structure
<i>nMaxSize</i>	the maximum size of the data structure, if the input size is larger than nMaxSize, the input will be truncated to nMaxSize

**7.19.3.2 DeleteHead()**

```
void Algorithm::DoublyLinkedList::DeleteHead ( )
```

Delete the head of the doubly linked list.

**7.19.3.3 DeleteMiddle()**

```
void Algorithm::DoublyLinkedList::DeleteMiddle (
    int index )
```

Delete an element at the middle of the doubly linked list.

**Parameters**

<i>index</i>	The index of the element to be deleted.
--------------	---

**Note**

This function is written as the visualization of deleting an element at the middle of the doubly linked list is different from deleting an element at the head or tail of the doubly linked list.

**7.19.3.4 DeleteTail()**

```
void Algorithm::DoublyLinkedList::DeleteTail ( )
```

Delete the tail of the doubly linked list.

**7.19.3.5 Empty()**

```
void Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLIAnimation >::Empty [virtual],
[inherited]
```

Initialize an empty data structure, and generate animation.

### 7.19.3.6 EmptyGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::←
EmptyGenerator [protected], [inherited]
```

Generate an empty input, this is used to generate the input for [Empty \(\)](#)

#### Returns

`std::vector< int >` the empty input

### 7.19.3.7 GenerateAnimation()

```
DLLAnimation Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::Generate←
Animation (
    float duration,
    int highlightLine,
    std::string actionDescription ) [protected], [virtual], [inherited]
```

Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)

#### Parameters

<code>duration</code>	the duration of the animation
<code>highlightLine</code>	the line to be highlighted in the code highlighter (if the line is -1, then no line will be highlighted)
<code>actionDescription</code>	the description of the action (which is used to display in the animation)

#### Returns

`AnimationState` the empty animation state (which currently only contains the metadata of the animation)

### 7.19.3.8 GenerateRelayoutAnimation()

```
void Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::GenerateRelayout←
Animation (
    Vector2 newPosition ) [inline], [protected], [virtual], [inherited]
```

Generate the relayout animation (which reposition the data structure to a new position on the screen)

#### Parameters

<code>newPosition</code>	the new position of the data structure
--------------------------	--

### 7.19.3.9 HighlightArrowBoth()

```
std::function< GUIVisualizer::DoublyLinkedList (GUIVisualizer::DoublyLinkedList, float, Vector2)
> Algorithm::DoublyLinkedList::HighlightArrowBoth (
    int index,
    bool drawVisualizer = true,
    bool reverse = false ) [private]
```

Generate a highlight arrow from the node at the given index to the previous node of the doubly linked list (in both direction).

#### Parameters

<i>index</i>	The index of the node.
<i>drawVisualizer</i>	Whether to draw the visualizer, this is set to true by default.
<i>reverse</i>	Whether to reverse the animation, this is set to false by default.

#### Returns

The animation of the highlight arrow.

### 7.19.3.10 HighlightArrowNext()

```
std::function< GUIVisualizer::DoublyLinkedList (GUIVisualizer::DoublyLinkedList, float, Vector2)
> Algorithm::DoublyLinkedList::HighlightArrowNext (
    int index,
    bool drawVisualizer = true,
    bool reverse = false ) [private]
```

Generate a highlight arrow from the node at the given index to the next node of the doubly linked list.

#### Parameters

<i>index</i>	The index of the node.
<i>drawVisualizer</i>	Whether to draw the visualizer, this is set to true by default.
<i>reverse</i>	Whether to reverse the animation, this is set to false by default.

#### Returns

The animation of the highlight arrow.

### 7.19.3.11 HighlightArrowPrev()

```
std::function< GUIVisualizer::DoublyLinkedList(GUIVisualizer::DoublyLinkedList, float, Vector2)
> Algorithm::DoublyLinkedList::HighlightArrowPrev (
    int index,
    bool drawVisualizer = true,
    bool reverse = false ) [private]
```

Generate a highlight arrow from the next node at the given index to the node at the given index of the doubly linked list.

#### Parameters

<i>index</i>	The index of the node.
<i>drawVisualizer</i>	Whether to draw the visualizer, this is set to true by default.
<i>reverse</i>	Whether to reverse the animation, this is set to false by default.

#### Returns

The animation of the highlight arrow.

### 7.19.3.12 InitAction()

```
void Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::InitAction (
    std::vector< std::string > code ) [protected], [virtual], [inherited]
```

Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.

#### Note

This function is used to clear the animation controller and the code highlighter, and then initialize a new instruction

### 7.19.3.13 InsertAfterTail()

```
void Algorithm::DoublyLinkedList::InsertAfterTail (
    int value )
```

Insert a new element after the tail of the doubly linked list.

#### Parameters

<i>value</i>	The value of the new element.
--------------	-------------------------------

### 7.19.3.14 InsertHead()

```
void Algorithm::DoublyLinkedList::InsertHead (
    int value )
```

Insert a new element before the head of the doubly linked list.

#### Parameters

<i>value</i>	The value of the new element.
--------------	-------------------------------

### 7.19.3.15 InsertMiddle()

```
void Algorithm::DoublyLinkedList::InsertMiddle (
    int index,
    int value )
```

Insert a new element at the middle of the doubly linked list.

#### Parameters

<i>index</i>	The index of the new element.
<i>value</i>	The value of the new element.

#### Note

This function is written as the visualization of inserting a new element at the middle of the doubly linked list is different from inserting a new element at the head or tail of the doubly linked list.

### 7.19.3.16 Random()

```
void Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::Random [virtual],
[inherited]
```

Initialize a data structure with random values input, and then generate animation.

### 7.19.3.17 RandomFixedSize()

```
void Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::RandomFixedSize (
    int N ) [virtual], [inherited]
```

Initialize a fixed size data structure with random values input, and then generate animation.

**Parameters**

<i>N</i>	the size of the data structure
----------	--------------------------------

**7.19.3.18 RandomFixedSizeGenerator()**

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::←
RandomFixedSizeGenerator (
    int nSize ) [protected], [inherited]
```

Generate a random input with fixed size, this is used to generate the input for [RandomFixedSize\(\)](#)

**Parameters**

<i>nSize</i>	the size of the input
--------------	-----------------------

**Returns**

```
std::vector< int > the random input with fixed size
```

**7.19.3.19 RandomGenerator()**

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::←
RandomGenerator [protected], [inherited]
```

Generate a random input, this is used to generate the input for [Random\(\)](#)

**Returns**

```
std::vector< int > the random input
```

**7.19.3.20 ReadFromExternalFile()**

```
void Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::ReadFromExternal←
File (
    std::string path ) [virtual], [inherited]
```

Initialize a data structure with input from external file (which is used to store the input), and then generate animation.

**Parameters**

<i>path</i>	the path to the external file
-------------	-------------------------------

### 7.19.3.21 ReadFromFileGenerator()

```
std::vector< int > Algorithm::Algorithm< GIVVisualizer::DoublyLinkedList , DLLAnimation >::←  
ReadFromFileGenerator (   
    std::string inputFile ) [protected], [inherited]
```

Parse the input from external file, this is used to generate the input for [ReadFromExternalFile \(\)](#)

#### Parameters

<i>inputFile</i>	the path to the external file
------------------	-------------------------------

#### Returns

```
std::vector< int > the input from external file
```

### 7.19.3.22 ResetVisualizer()

```
void Algorithm::DoublyLinkedList::ResetVisualizer ( ) [private]
```

Reset the visualizer of the doubly linked list algorithm.

#### Note

This function is used to reset all of the effects that are only used to visualize the algorithm (i.e highlight the elements that are currently iterated).

### 7.19.3.23 Search()

```
void Algorithm::DoublyLinkedList::Search (   
    int value )
```

Search for an element in the doubly linked list.

#### Parameters

<i>value</i>	The value of the element to be searched.
--------------	--

### 7.19.3.24 size()

```
std::size_t Algorithm::DoublyLinkedList::size ( ) const
```

Return the size of the doubly linked list.

#### Returns

The size of the doubly linked list.

### 7.19.3.25 Update()

```
void Algorithm::DoublyLinkedList::Update (
    int index,
    int value )
```

Update the value of an element at the head of the doubly linked list.

#### Parameters

<code>value</code>	The new value of the element.
--------------------	-------------------------------

### 7.19.3.26 UserDefined()

```
void Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::UserDefined (
    std::string input ) [virtual], [inherited]
```

Initialize a data structure with input from user, and then generate animation.

#### Parameters

<code>input</code>	the input from user
--------------------	---------------------

### 7.19.3.27 UserDefinedGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::UserDefinedGenerator (
    std::string input ) [protected], [inherited]
```

Generate an input from user, this is used to generate the input for [UserDefined\(\)](#)

**Parameters**

<i>input</i>	the input from user
--------------	---------------------

**Returns**

std::vector< int > the input from user

## 7.19.4 Member Data Documentation

### 7.19.4.1 animController

```
Animation::AnimationController<DLLEnimation >::Ptr Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLEnimation >::animController [protected], [inherited]
```

Animation controller for the algorithm (which is used to control the animation generated by the algorithm)

**Note**

This [Algorithm](#) class is mainly use the animation controller to add animation for the algorithm

### 7.19.4.2 codeHighlighter

```
GUIComponent::CodeHighlighter::Ptr Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLEnimation >::codeHighlighter [protected], [inherited]
```

Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)

### 7.19.4.3 maxN

```
constexpr int Algorithm::DoublyLinkedList::maxN = 16 [static], [constexpr]
```

The maximum number of elements that the doubly linked list can hold.

#### 7.19.4.4 visualizer

```
GUIVisualizer::DoublyLinkedList Algorithm::Algorithm< GUIVisualizer::DoublyLinkedList , DLLAnimation >::visualizer [protected], [inherited]
```

Visualizer for the algorithm (which is used to draw animation generated by the algorithm)

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

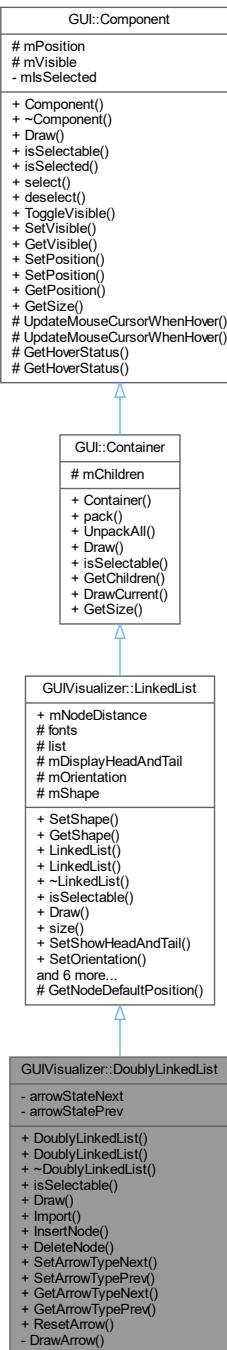
- src/Algorithms/LinkedList/DoublyLinkedList.hpp
- src/Algorithms/LinkedList/DoublyLinkedList.cpp

## 7.20 GUIVisualizer::DoublyLinkedList Class Reference

The doubly linked list visualization.

```
#include <DoublyLinkedList.hpp>
```

Inheritance diagram for GUIVisualizer::DoublyLinkedList:



## Public Types

- enum `ArrowType` {
 `Default` , `Hidden` , `Active` , `Skip` ,
 `ArrowTypeCount` }
- The type of the arrow.*
- enum `Orientation` { `Horizontal` , `Vertical` , `OrientationCount` }

- The orientation of the linked list.*
- `typedef std::shared_ptr< Container > Ptr`  
*The shared pointer to the container.*
- ## Public Member Functions
- `DoublyLinkedList ()`  
*Construct a new Doubly Linked List object.*
  - `DoublyLinkedList (FontHolder *fonts)`  
*Construct a new Doubly Linked List object.*
  - `~DoublyLinkedList ()`  
*Destroy the Doubly Linked List object.*
  - `bool IsSelectable () const`  
*Check if the container is selectable.*
  - `void Draw (Vector2 base=(Vector2){0, 0}, float t=1.0f, bool init=false)`  
*Draw the doubly linked list visualization.*
  - `void Import (std::vector< int > nodes)`  
*Initialize the doubly linked list visualization with the given nodes.*
  - `void InsertNode (std::size_t index, Node node, bool rePosition=true)`  
*Insert a node into the doubly linked list visualization.*
  - `void DeleteNode (std::size_t index, bool rePosition=true)`  
*Delete a node from the doubly linked list visualization.*
  - `void SetArrowTypeNext (std::size_t index, ArrowType type)`
  - `void SetArrowTypePrev (std::size_t index, ArrowType type)`
  - `ArrowType GetArrowTypeNext (std::size_t index)`
  - `ArrowType GetArrowTypePrev (std::size_t index)`
  - `void ResetArrow ()`
  - `void SetShape (Node::Shape shape)`  
*Set the shape of the node.*
  - `Node::Shape GetShape () const`  
*Get the shape of the node.*
  - `virtual void Draw (Vector2 base=(Vector2){0, 0})`  
*Draw the container.*
  - `virtual std::size_t size () const`  
*Get the size of the linked list visualization.*
  - `virtual void SetShowHeadAndTail (bool show)`
  - `virtual void SetOrientation (Orientation orientation)`  
*Set the orientation of the linked list visualization.*
  - `virtual std::vector< Node > & GetList ()`
  - `virtual Node GenerateNode (int value)`
  - `virtual void Relayout ()`  
*Relayout the linked list visualization.*
  - `void pack (Component::Ptr component)`  
*Pack a component into the container.*
  - `void UnpackAll ()`  
*Unpack all components from the container.*
  - `Core::Deque< Component::Ptr > GetChildren ()`  
*Get the pack components of the container.*
  - `virtual void DrawCurrent (Vector2 base)`
  - `virtual Vector2 GetSize ()`  
*Get the size of the container.*

- bool `isSelected () const`  
*Check if the component is selected.*
- virtual void `select ()`  
*Select the component.*
- virtual void `deselect ()`  
*Deselect the component.*
- virtual void `ToggleVisible ()`  
*Toggle the visibility of the component.*
- virtual void `SetVisible (bool visible)`  
*Set the visibility of the component.*
- virtual bool `GetVisible ()`  
*Get the visibility of the component.*
- void `SetPosition (float x, float y)`  
*Set the position of the component.*
- void `SetPosition (Vector2 position)`  
*Set the position of the component.*
- Vector2 `GetPosition ()`  
*Get the position of the component.*

## Static Public Attributes

- static constexpr float `mNodeDistance = 20`  
*The distance between the `GUI` nodes.*

## Protected Member Functions

- Vector2 `GetNodeDefaultPosition (std::size_t index)`
- virtual void `UpdateMouseCursorWhenHover (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
*Update the mouse cursor when hovering.*
- virtual void `UpdateMouseCursorWhenHover (Rectangle bound, bool hover, bool noHover)`  
*Update the mouse cursor when hovering.*
- virtual bool `GetHoverStatus (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
*Get the hover status of the component.*
- virtual bool `GetHoverStatus (Rectangle bound, bool hover, bool noHover)`  
*Get the hover status of the component.*

## Protected Attributes

- `FontHolder * fonts`
- `std::vector< Node > list`
- bool `mDisplayHeadAndTail`
- `Orientation mOrientation = Orientation::Horizontal`  
*The orientation of the linked list, by default it is horizontal.*
- `Node::Shape mShape = Node::Shape::Circle`  
*The shape of the node, by default it is a circle.*
- `Core::Deque< Component::Ptr > mChildren`
- `Vector2 mPosition`  
*The position of the component.*
- bool `mVisible`  
*The visibility of the component.*

## Private Member Functions

- void [DrawArrow](#) (Vector2 base, float t)

## Private Attributes

- std::vector< [ArrowType](#) > arrowStateNext
- std::vector< [ArrowType](#) > arrowStatePrev
- bool [mIsSelected](#)

*The selected status of the component.*

### 7.20.1 Detailed Description

The doubly linked list visualization.

The doubly linked list visualization is used to visualize the doubly linked list.

### 7.20.2 Member Typedef Documentation

#### 7.20.2.1 Ptr

```
typedef std::shared_ptr< Container > GUI::Container::Ptr [inherited]
```

The shared pointer to the container.

The [GUI](#) container is commonly used by multiple components, so a shared pointer is used.

See also

[std::shared\\_ptr](#)

### 7.20.3 Member Enumeration Documentation

#### 7.20.3.1 ArrowType

```
enum GUIVisualizer::LinkedList::ArrowType [inherited]
```

The type of the arrow.

The type of the arrow is used to determine the color of the arrow.

**Enumerator**

Default	
Hidden	
Active	
Skip	
ArrowTypeCount	

**7.20.3.2 Orientation**

```
enum GUIVisualizer::LinkedList::Orientation [inherited]
```

The orientation of the linked list.

The orientation of the linked list is used to determine the orientation of the linked list.

**Enumerator**

Horizontal	
Vertical	
OrientationCount	

**7.20.4 Constructor & Destructor Documentation****7.20.4.1 DoublyLinkedList() [1/2]**

```
GUIVisualizer::DoublyLinkedList::DoublyLinkedList ( )
```

Construct a new Doubly Linked List object.

**7.20.4.2 DoublyLinkedList() [2/2]**

```
GUIVisualizer::DoublyLinkedList::DoublyLinkedList (
    FontHolder * fonts )
```

Construct a new Doubly Linked List object.

#### 7.20.4.3 ~DoublyLinkedList()

```
GUIVisualizer::DoublyLinkedList::~DoublyLinkedList ( )
```

Destroy the Doubly Linked List object.

### 7.20.5 Member Function Documentation

#### 7.20.5.1 DeleteNode()

```
void GUIVisualizer::DoublyLinkedList::DeleteNode (   
    std::size_t index,  
    bool rePosition = true ) [virtual]
```

Delete a node from the doubly linked list visualization.

##### Parameters

<i>index</i>	The index to delete the node.
<i>rePosition</i>	Whether to reposition the whole doubly linked list after deletion.

##### See also

[LinkedList::DeleteNode](#)

Reimplemented from [GUIVisualizer::LinkedList](#).

#### 7.20.5.2 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

#### 7.20.5.3 Draw() [1/2]

```
void GUI::Container::Draw (   
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Draw the container.

This function draws the container.

**Parameters**

<i>base</i>	The base position of the container.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#).

**7.20.5.4 Draw() [2/2]**

```
void GUIVisualizer::DoublyLinkedList::Draw (
    Vector2 base = (Vector2){0, 0},
    float t = 1.0f,
    bool init = false )  [virtual]
```

Draw the doubly linked list visualization.

This function draws the doubly linked list visualization.

**Parameters**

<i>base</i>	The base position of the doubly linked list visualization.
<i>t</i>	The time delta.
<i>init</i>	True if the doubly linked list visualization is initializing.

Implements [GUIVisualizer::LinkedList](#).

**7.20.5.5 DrawArrow()**

```
void GUIVisualizer::DoublyLinkedList::DrawArrow (
    Vector2 base,
    float t )  [private]
```

**7.20.5.6 DrawCurrent()**

```
void GUI::Container::DrawCurrent (
    Vector2 base = (Vector2){0, 0} )  [virtual], [inherited]
```

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

### 7.20.5.7 GenerateNode()

```
GUIVisualizer::Node GUIVisualizer::LinkedList::GenerateNode (
    int value ) [virtual], [inherited]
```

### 7.20.5.8 GetArrowTypeNext()

```
GUIVisualizer::DoublyLinkedList::ArrowType GUIVisualizer::DoublyLinkedList::GetArrowTypeNext (
    std::size_t index )
```

### 7.20.5.9 GetArrowTypePrev()

```
GUIVisualizer::DoublyLinkedList::ArrowType GUIVisualizer::DoublyLinkedList::GetArrowTypePrev (
    std::size_t index )
```

### 7.20.5.10 GetChildren()

```
Core::Deque< GUI::Component::Ptr > GUI::Container::GetChildren () [inherited]
```

Get the pack components of the container.

This function returns the all of the pack components of the container.

#### Returns

The children of the container.

### 7.20.5.11 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

#### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

### 7.20.5.12 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

#### Parameters

<code>bounds</code>	The list of bound of the component.
---------------------	-------------------------------------

### 7.20.5.13 GetList()

```
std::vector< GUIVisualizer::Node > & GUIVisualizer::LinkedList::GetList () [virtual], [inherited]
```

### 7.20.5.14 GetNodeDefaultPosition()

```
Vector2 GUIVisualizer::LinkedList::GetNodeDefaultPosition (
    std::size_t index ) [protected], [inherited]
```

### 7.20.5.15GetPosition()

```
Vector2 GUI::Component::GetPosition () [inherited]
```

Get the position of the component.

This function gets the position of the component.

#### Returns

The position of the component.

### 7.20.5.16 GetShape()

```
GUIVisualizer::Node::Shape GUIVisualizer::LinkedList::GetShape ( ) const [inherited]
```

Get the shape of the node.

### 7.20.5.17 GetSize()

```
Vector2 GUI::Container::GetSize ( ) [virtual], [inherited]
```

Get the size of the container.

This function returns the size of the container.

#### Returns

The size of the container.

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#), and [GUIComponent::OptionInputField](#).

### 7.20.5.18 GetVisible()

```
bool GUI::Component::GetVisible ( ) [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

#### Returns

The visibility of the component.

### 7.20.5.19 Import()

```
void GUIVisualizer::DoublyLinkedList::Import ( std::vector< int > nodes ) [virtual]
```

Initialize the doubly linked list visualization with the given nodes.

#### Parameters

<i>nodes</i>	The nodes to initialize the doubly linked list visualization with.
--------------	--

See also

[LinkedList::Import](#)

Reimplemented from [GUIVisualizer::LinkedList](#).

#### 7.20.5.20 InsertNode()

```
void GUIVisualizer::DoublyLinkedList::InsertNode (
    std::size_t index,
    GUIVisualizer::Node node,
    bool rePosition = true ) [virtual]
```

Insert a node into the doubly linked list visualization.

Parameters

<i>index</i>	The index to insert the node.
<i>node</i>	The node to insert.
<i>rePosition</i>	Whether to reposition the whole doubly linked list after insertion.

See also

[LinkedList::InsertNode](#)

Reimplemented from [GUIVisualizer::LinkedList](#).

#### 7.20.5.21 isSelectable()

```
bool GUIVisualizer::DoublyLinkedList::isSelectable () const [virtual]
```

Check if the container is selectable.

**Deprecated** This function is deprecated.

This function checks if the container is selectable.

Returns

True if the container is selectable.

Reimplemented from [GUI::Container](#).

### 7.20.5.22 isSelected()

```
bool GUI::Component::isSelected ( ) const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

#### Returns

True if the component is selected.

### 7.20.5.23 pack()

```
void GUI::Container::pack (
    Component::Ptr component ) [inherited]
```

Pack a component into the container.

This function packs a component into the container.

#### Parameters

<code>component</code>	The component to pack.
------------------------	------------------------

### 7.20.5.24 Relayout()

```
void GUIVisualizer::LinkedList::Relayout ( ) [virtual], [inherited]
```

relayout the linked list visualization.

### 7.20.5.25 ResetArrow()

```
void GUIVisualizer::DoublyLinkedList::ResetArrow ( )
```

### 7.20.5.26 select()

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

### 7.20.5.27 SetArrowTypeNext()

```
void GUVISUALIZER::DoublyLinkedList::SetArrowTypeNext (
    std::size_t index,
    ArrowType type )
```

### 7.20.5.28 SetArrowTypePrev()

```
void GUVISUALIZER::DoublyLinkedList::SetArrowTypePrev (
    std::size_t index,
    ArrowType type )
```

### 7.20.5.29 SetOrientation()

```
void GUVISUALIZER::LinkedList::SetOrientation (
    Orientation orientation ) [virtual], [inherited]
```

Set the orientation of the linked list visualization.

#### Parameters

<i>orientation</i>	The orientation of the linked list visualization.
--------------------	---

### 7.20.5.30 SetPosition() [1/2]

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

**7.20.5.31 SetPosition() [2/2]**

```
void GUI::Component::SetPosition (
    Vector2 position ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>position</i>	The position of the component.
-----------------	--------------------------------

**7.20.5.32 SetShape()**

```
void GUIVisualizer::LinkedList::SetShape (
    Node::Shape shape ) [inherited]
```

Set the shape of the node.

**7.20.5.33 SetShowHeadAndTail()**

```
void GUIVisualizer::LinkedList::SetShowHeadAndTail (
    bool show ) [virtual], [inherited]
```

**7.20.5.34 SetVisible()**

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

**Parameters**

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

**7.20.5.35 size()**

```
std::size_t GUIVisualizer::LinkedList::size () const [virtual], [inherited]
```

Get the size of the linked list visualization.

**Returns**

The size of the linked list visualization.

**7.20.5.36 ToggleVisible()**

```
void GUI::Component::ToggleVisible () [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

**7.20.5.37 UnpackAll()**

```
void GUI::Container::UnpackAll () [inherited]
```

Unpack all components from the container.

This function unpacks all components from the container.

**See also**

[pack](#)

**7.20.5.38 UpdateMouseCursorWhenHover() [1/2]**

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

**7.20.5.39 UpdateMouseCursorWhenHover() [2/2]**

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.20.6 Member Data Documentation****7.20.6.1 arrowStateNext**

```
std::vector< ArrowType > GUIVisualizer::DoublyLinkedList::arrowStateNext [private]
```

**7.20.6.2 arrowStatePrev**

```
std::vector< ArrowType > GUIVisualizer::DoublyLinkedList::arrowStatePrev [private]
```

**7.20.6.3 fonts**

```
FontHolder* GUIVisualizer::LinkedList::fonts [protected], [inherited]
```

#### 7.20.6.4 list

```
std::vector< Node > GUIVisualizer::LinkedList::list [protected], [inherited]
```

#### 7.20.6.5 mChildren

```
Core::Deque< Component::Ptr > GUI::Container::mChildren [protected], [inherited]
```

#### 7.20.6.6 mDisplayHeadAndTail

```
bool GUIVisualizer::LinkedList::mDisplayHeadAndTail [protected], [inherited]
```

#### 7.20.6.7 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

#### 7.20.6.8 mNodeDistance

```
constexpr float GUIVisualizer::LinkedList::mNodeDistance = 20 [static], [constexpr], [inherited]
```

The distance between the [GUI](#) nodes.

#### 7.20.6.9 mOrientation

```
Orientation GUIVisualizer::LinkedList::mOrientation = Orientation::Horizontal [protected], [inherited]
```

The orientation of the linked list, by default it is horizontal.

### 7.20.6.10 mPosition

`Vector2 GUI::Component::mPosition [protected], [inherited]`

The position of the component.

This variable stores the position of the component.

### 7.20.6.11 mShape

`Node::Shape GUIVisualizer::LinkedList::mShape = Node::Shape::Circle [protected], [inherited]`

The shape of the node, by default it is a circle.

### 7.20.6.12 mVisible

`bool GUI::Component::mVisible [protected], [inherited]`

The visibility of the component.

This variable stores the visibility of the component.

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

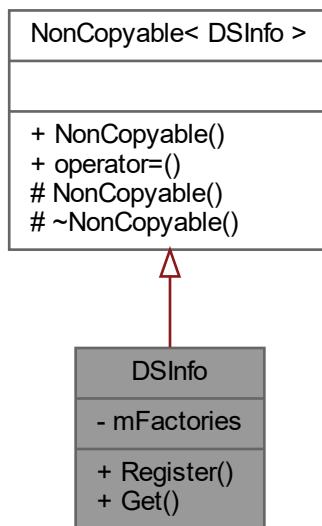
- `src/Components/Visualization/DoublyLinkedList.hpp`
- `src/Components/Visualization/DoublyLinkedList.cpp`

## 7.21 DSInfo Class Reference

The data structure info class that is used to store information about the data structures.

`#include <DSInfo.hpp>`

Inheritance diagram for DSInfo:



## Classes

- struct [Info](#)

*The info struct that is used to store information about the data structures.*

## Public Member Functions

- void [Register \(DataStructures::ID, Info info\)](#)  
*This function is used to register the data structure info.*
- [Info Get \(DataStructures::ID id\) const](#)  
*This function is used to retrieve the data structure info.*

## Private Attributes

- std::map< [DataStructures::ID](#), [Info](#) > [mFactories](#)  
*The factory map that is used to store the factories.*

### 7.21.1 Detailed Description

The data structure info class that is used to store information about the data structures.

The data structure info class is used to store information about the data structures. Also, this class is using the flyweight pattern to store the information about the data structures.

See also

[DataStructures::ID](#)  
[States::ID](#)  
[Category::ID](#)  
[Textures::ID](#)  
[NonCopyable](#)  
[Info](#)  
[Register](#)

### 7.21.2 Member Function Documentation

#### 7.21.2.1 Get()

```
DSInfo::Info DSInfo::Get (  
    DataStructures::ID id ) const
```

This function is used to retrieve the data structure info.

This function is used to retrieve the data structure info.

**Parameters**

<i>id</i>	The data structure ID.
-----------	------------------------

**Returns**

The data structure info.

### 7.21.2.2 Register()

```
void DSInfo::Register (
    DataStructures::ID id,
    Info info )
```

This function is used to register the data structure info.

This function is used to register the data structure info.

**Parameters**

<i>id</i>	The data structure ID.
<i>info</i>	The data structure info.

## 7.21.3 Member Data Documentation

### 7.21.3.1 mFactories

```
std::map< DataStructures::ID, Info > DSInfo::mFactories [private]
```

The factory map that is used to store the factories.

The factory map is used to store the factories. The factories are used to create the data structures.

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

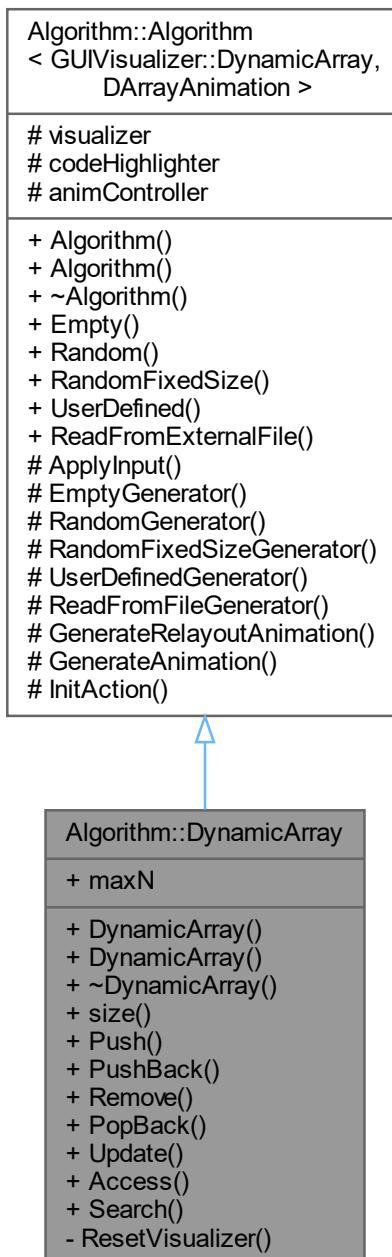
- src/Identifiers/DSInfo.hpp
- src/Identifiers/DSInfo.cpp

## 7.22 Algorithm::DynamicArray Class Reference

The algorithm class that is used to generate step-by-step instructions for the visualization of the dynamic array.

```
#include <DynamicArray.hpp>
```

Inheritance diagram for Algorithm::DynamicArray:



## Public Member Functions

- **DynamicArray ()**  
*The constructor of the dynamic array algorithm.*
- **DynamicArray (GUIComponent::CodeHighlighter::Ptr codeHighlighter, DArrayAnimationController::Ptr animController, FontHolder \*fonts)**  
*The constructor of the dynamic array algorithm.*
- **~DynamicArray ()**  
*The destructor of the dynamic array algorithm.*
- **std::size\_t size () const**  
*Return the size of the dynamic array.*
- **void Push (int index, int value)**  
*Push a new element to the dynamic array.*
- **void PushBack (int value)**  
*Push a new element to the back of the dynamic array.*
- **void Remove (int index)**  
*Remove an element from the dynamic array.*
- **void PopBack ()**  
*Remove an element from the back of the dynamic array.*
- **void Update (int index, int value)**  
*Update the value of an element in the dynamic array.*
- **void Access (int index)**  
*Access an element in the dynamic array.*
- **void Search (int value)**  
*Search for an element in the dynamic array.*
- **virtual void Empty ()**  
*Initialize an empty data structure, and generate animation.*
- **virtual void Random ()**  
*Initialize a data structure with random values input, and then generate animation.*
- **virtual void RandomFixedSize (int N)**  
*Initialize a fixed size data structure with random values input, and then generate animation.*
- **virtual void UserDefined (std::string input)**  
*Initialize a data structure with input from user, and then generate animation.*
- **virtual void ReadFromExternalFile (std::string path)**  
*Initialize a data structure with input from external file (which is used to store the input), and then generate animation.*

## Static Public Attributes

- static constexpr int **maxN** = 16

## Protected Member Functions

- **virtual void ApplyInput (std::vector< int > input, std::size\_t nMaxSize=10)**  
*Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an std::vector< int >) to the data structure, used by other initialization functions.*
- **std::vector< int > EmptyGenerator ()**  
*Generate an empty input, this is used to generate the input for `Empty ()`*
- **std::vector< int > RandomGenerator ()**  
*Generate a random input, this is used to generate the input for `Random ()`*
- **std::vector< int > RandomFixedSizeGenerator (int nSize)**

- std::vector< int > **UserDefinedGenerator** (std::string input)
 

*Generate a random input with fixed size, this is used to generate the input for RandomFixedSize ()*
- std::vector< int > **ReadFromFileGenerator** (std::string inputFile)
 

*Parse the input from external file, this is used to generate the input for ReadFromExternalFile ()*
- virtual void **GenerateRelayoutAnimation** (Vector2 newPosition)
 

*Generate the relayout animation (which reposition the data structure to a new position on the screen)*
- virtual **DArrayAnimation GenerateAnimation** (float duration, int highlightLine, std::string actionDescription)
 

*Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)*
- virtual void **InitAction** (std::vector< std::string > code)
 

*Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.*

## Protected Attributes

- **GUIVisualizer::DynamicArray visualizer**

*Visualizer for the algorithm (which is used to draw animation generated by the algorithm)*
- **GUIComponent::CodeHighlighter::Ptr codeHighlighter**

*Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)*
- **Animation::AnimationController< DArrayAnimation >::Ptr animController**

*Animation controller for the algorithm (which is used to control the animation generated by the algorithm)*

## Private Member Functions

- void **ResetVisualizer** ()
 

*Reset the visualizer of the dynamic array.*

### 7.22.1 Detailed Description

The algorithm class that is used to generate step-by-step instructions for the visualization of the dynamic array.

### 7.22.2 Constructor & Destructor Documentation

#### 7.22.2.1 DynamicArray() [1/2]

```
Algorithm::DynamicArray::DynamicArray ( )
```

The constructor of the dynamic array algorithm.

#### 7.22.2.2 DynamicArray() [2/2]

```
Algorithm::DynamicArray::DynamicArray (
    GUIComponent::CodeHighlighter::Ptr codeHighlighter,
    DArrayAnimationController::Ptr animController,
    FontHolder * fonts )
```

The constructor of the dynamic array algorithm.

**Parameters**

<i>codeHighlighter</i>	The code highlighter of the dynamic array algorithm.
<i>animController</i>	The animation controller of the dynamic array algorithm.
<i>fonts</i>	The fonts of the dynamic array algorithm.

**7.22.2.3 ~DynamicArray()**

```
Algorithm::DynamicArray::~DynamicArray ( )
```

The destructor of the dynamic array algorithm.

**Note**

This destructor is not virtual because this class is not designed to be inherited.

**7.22.3 Member Function Documentation****7.22.3.1 Access()**

```
void Algorithm::DynamicArray::Access (
    int index )
```

Access an element in the dynamic array.

**Parameters**

<i>index</i>	The index of the element to be accessed.
--------------	--

**7.22.3.2 ApplyInput()**

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::ApplyInput (
    std::vector< int > input,
    std::size_t nMaxSize = 10 ) [protected], [virtual], [inherited]
```

Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an `std::vector< int >`) to the data structure, used by other initialization functions.

**Parameters**

<i>input</i>	the input (in <code>std::vector&lt; int &gt;</code> ) to be applied to the data structure
<i>nMaxSize</i>	the maximum size of the data structure, if the input size is larger than <i>nMaxSize</i> , the input will be truncated to <i>nMaxSize</i>
<small>Generated by Doxygen</small>	

Reimplemented in [Algorithm::StaticArray](#).

### 7.22.3.3 Empty()

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::Empty [virtual],  
[inherited]
```

Initialize an empty data structure, and generate animation.

### 7.22.3.4 EmptyGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::←  
EmptyGenerator [protected], [inherited]
```

Generate an empty input, this is used to generate the input for [Empty \(\)](#)

#### Returns

```
std::vector< int > the empty input
```

### 7.22.3.5 GenerateAnimation()

```
DArrayAnimation Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::←  
GenerateAnimation (   
    float duration,  
    int highlightLine,  
    std::string actionDescription ) [protected], [virtual], [inherited]
```

Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)

#### Parameters

<i>duration</i>	the duration of the animation
<i>highlightLine</i>	the line to be highlighted in the code highlighter (if the line is -1, then no line will be highlighted)
<i>actionDescription</i>	the description of the action (which is used to display in the animation)

#### Returns

```
AnimationState the empty animation state (which currently only contains the metadata of the animation)
```

### 7.22.3.6 GenerateRelayoutAnimation()

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::GenerateRelayoutAnimation ( Vector2 newPosition ) [inline], [protected], [virtual], [inherited]
```

Generate the relayout animation (which reposition the data structure to a new position on the screen)

#### Parameters

<i>newPosition</i>	the new position of the data structure
--------------------	--

### 7.22.3.7 InitAction()

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::InitAction ( std::vector< std::string > code ) [protected], [virtual], [inherited]
```

Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.

#### Note

This function is used to clear the animation controller and the code highlighter, and then initialize a new instruction

### 7.22.3.8 PopBack()

```
void Algorithm::DynamicArray::PopBack ( )
```

Remove an element from the back of the dynamic array.

#### Note

This function will not change the capacity of the dynamic array even if the size of the dynamic array is much smaller than the half of the capacity.

### 7.22.3.9 Push()

```
void Algorithm::DynamicArray::Push ( int index, int value )
```

Push a new element to the dynamic array.

**Parameters**

<i>index</i>	The index of the new element.
<i>value</i>	The value of the new element.

**Note**

If the size of the dynamic array is equal to the capacity (which there is not reserved space for new elements), the capacity of the dynamic array will be doubled. We do this by initialize a new dynamic array with the new capacity, and copy all the elements from the old dynamic array to the new dynamic array.

**7.22.3.10 PushBack()**

```
void Algorithm::DynamicArray::PushBack (
    int value )
```

Push a new element to the back of the dynamic array.

**Parameters**

<i>value</i>	The value of the new element.
--------------	-------------------------------

**Note**

If the size of the dynamic array is equal to the capacity (which there is not reserved space for new elements), the capacity of the dynamic array will be doubled. We do this by initialize a new dynamic array with the new capacity, and copy all the elements from the old dynamic array to the new dynamic array.

**7.22.3.11 Random()**

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::Random [virtual],
[inherited]
```

Initialize a data structure with random values input, and then generate animation.

**7.22.3.12 RandomFixedSize()**

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::RandomFixedSize (
    int N ) [virtual], [inherited]
```

Initialize a fixed size data structure with random values input, and then generate animation.

**Parameters**

<i>N</i>	the size of the data structure
----------	--------------------------------

**7.22.3.13 RandomFixedSizeGenerator()**

```
std::vector< int > Algorithm::Algorithm< GIVisualizer::DynamicArray , DArrayAnimation >::←
RandomFixedSizeGenerator (
    int nSize ) [protected], [inherited]
```

Generate a random input with fixed size, this is used to generate the input for [RandomFixedSize\(\)](#)

**Parameters**

<i>nSize</i>	the size of the input
--------------	-----------------------

**Returns**

```
std::vector< int > the random input with fixed size
```

**7.22.3.14 RandomGenerator()**

```
std::vector< int > Algorithm::Algorithm< GIVisualizer::DynamicArray , DArrayAnimation >::←
RandomGenerator [protected], [inherited]
```

Generate a random input, this is used to generate the input for [Random\(\)](#)

**Returns**

```
std::vector< int > the random input
```

**7.22.3.15 ReadFromExternalFile()**

```
void Algorithm::Algorithm< GIVisualizer::DynamicArray , DArrayAnimation >::ReadFromExternal←
File (
    std::string path ) [virtual], [inherited]
```

Initialize a data structure with input from external file (which is used to store the input), and then generate animation.

**Parameters**

<i>path</i>	the path to the external file
-------------	-------------------------------

### 7.22.3.16 ReadFromFileGenerator()

```
std::vector< int > Algorithm::Algorithm< GIVVisualizer::DynamicArray , DArrayAnimation >::←
ReadFromFileGenerator (
    std::string inputFile ) [protected], [inherited]
```

Parse the input from external file, this is used to generate the input for [ReadFromExternalFile\(\)](#)

#### Parameters

<i>inputFile</i>	the path to the external file
------------------	-------------------------------

#### Returns

```
std::vector< int > the input from external file
```

### 7.22.3.17 Remove()

```
void Algorithm::DynamicArray::Remove (
    int index )
```

Remove an element from the dynamic array.

#### Parameters

<i>index</i>	The index of the element to be removed.
--------------	---

#### Note

This function will not change the capacity of the dynamic array even if the size of the dynamic array is much smaller than the half of the capacity.

### 7.22.3.18 ResetVisualizer()

```
void Algorithm::DynamicArray::ResetVisualizer ( ) [private]
```

Reset the visualizer of the dynamic array.

#### Note

This function is used to reset all of the effects that are only used to visualize the algorithm (i.e highlight the elements that are currently iterated).

### 7.22.3.19 Search()

```
void Algorithm::DynamicArray::Search (
    int value )
```

Search for an element in the dynamic array.

#### Parameters

<i>value</i>	The value of the element to be searched.
--------------	--

### 7.22.3.20 size()

```
std::size_t Algorithm::DynamicArray::size ( ) const
```

Return the size of the dynamic array.

#### Returns

The size of the dynamic array.

### 7.22.3.21 Update()

```
void Algorithm::DynamicArray::Update (
    int index,
    int value )
```

Update the value of an element in the dynamic array.

#### Parameters

<i>index</i>	The index of the element to be updated.
<i>value</i>	The new value of the element.

### 7.22.3.22 UserDefined()

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::UserDefined (
    std::string input ) [virtual], [inherited]
```

Initialize a data structure with input from user, and then generate animation.

**Parameters**

<i>input</i>	the input from user
--------------	---------------------

**7.22.3.23 UserDefinedGenerator()**

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::←
UserDefinedGenerator (
    std::string input ) [protected], [inherited]
```

Generate an input from user, this is used to generate the input for [UserDefined\(\)](#)

**Parameters**

<i>input</i>	the input from user
--------------	---------------------

**Returns**

```
std::vector< int > the input from user
```

**7.22.4 Member Data Documentation****7.22.4.1 animController**

```
Animation::AnimationController<DArrayAnimation >::Ptr Algorithm::Algorithm< GUIVisualizer::DynamicArray ,
, DArrayAnimation >::animController [protected], [inherited]
```

[Animation](#) controller for the algorithm (which is used to control the animation generated by the algorithm)

**Note**

This [Algorithm](#) class is mainly use the animation controller to add animation for the algorithm

**7.22.4.2 codeHighlighter**

```
GUIComponent::CodeHighlighter::Ptr Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation
>::codeHighlighter [protected], [inherited]
```

Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)

#### 7.22.4.3 maxN

```
constexpr int Algorithm::DynamicArray::maxN = 16 [static], [constexpr]
```

#### 7.22.4.4 visualizer

```
GUIVisualizer::DynamicArray Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::visualizer [protected], [inherited]
```

Visualizer for the algorithm (which is used to draw animation generated by the algorithm)

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

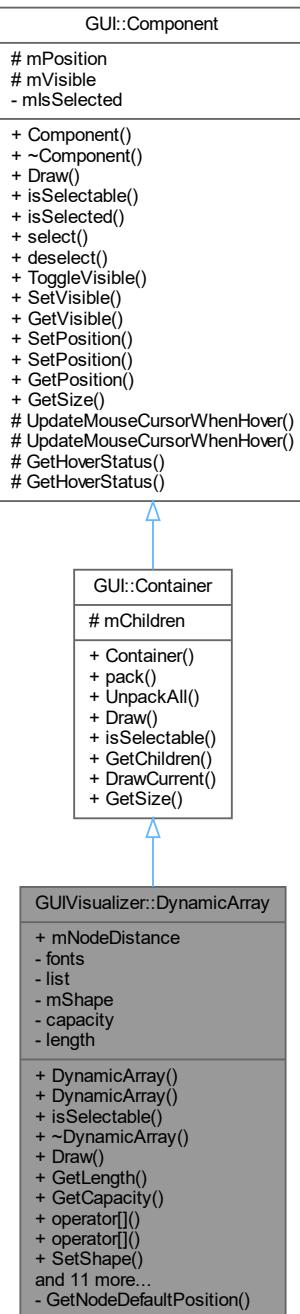
- src/Algorithms/Array/[DynamicArray.hpp](#)
- src/Algorithms/Array/[DynamicArray.cpp](#)

## 7.23 GUIVisualizer::DynamicArray Class Reference

The base class for the dynamic array visualization. This class provides the basic functionality for the dynamic array visualization.

```
#include <DynamicArray.hpp>
```

Inheritance diagram for GUIVisualizer::DynamicArray:



## Public Types

- `typedef std::shared_ptr< Container > Ptr`
- The shared pointer to the container.*

## Public Member Functions

- **DynamicArray ()**  
*The default constructor of the dynamic array visualization.*
- **DynamicArray (FontHolder \*fonts)**  
*The constructor of the dynamic array visualization.*
- **bool isSelectable () const**  
*Check if the container is selectable.*
- **~DynamicArray ()**  
*The destructor of the dynamic array visualization.*
- **void Draw (Vector2 base=(Vector2){0, 0}, float t=1.0f, bool init=false)**  
*Draw the dynamic array visualization.*
- **std::size\_t GetLength () const**  
*Get the length of the dynamic array visualization.*
- **std::size\_t GetCapacity () const**  
*Get the capacity of the dynamic array visualization.*
- **Node & operator[] (std::size\_t index)**  
*Get the node at the given index.*
- **const Node & operator[] (std::size\_t index) const**  
*Get the node at the given index.*
- **void SetShape (Node::Shape shape)**  
*Set the shape of the dynamic array visualization.*
- **Node::Shape GetShape () const**  
*Get the shape of the dynamic array visualization.*
- **void Reserve (std::size\_t size)**  
*Reserve the given size of the dynamic array visualization (more specific, adjust the capacity of the array).*
- **void Resize (std::size\_t size)**  
*Resize the dynamic array visualization.*
- **void Clear ()**  
*Clear the dynamic array visualization.*
- **std::vector< Node > & GetList ()**  
*Get the list of nodes.*
- **Node GenerateNode (int value)**
- **void Import (std::vector< int > nodes)**  
*Initialize the dynamic array visualization with the given nodes.*
- **void InsertNode (std::size\_t index, Node node, bool rePosition=true)**  
*Insert the given node at the given index.*
- **void DeleteNode (std::size\_t index, bool rePosition=true)**  
*Delete the node at the given index.*
- **void Relayout ()**  
*Relayout the array visualization.*
- **std::size\_t GetCapacityFromLength (std::size\_t length)**
- **void pack (Component::Ptr component)**  
*Pack a component into the container.*
- **void UnpackAll ()**  
*Unpack all components from the container.*
- **virtual void Draw (Vector2 base=(Vector2){0, 0})**  
*Draw the container.*
- **Core::Deque< Component::Ptr > GetChildren ()**  
*Get the pack components of the container.*
- **virtual void DrawCurrent (Vector2 base)**

- virtual Vector2 **GetSize** ()
 

*Get the size of the container.*
- bool **isSelected** () const
 

*Check if the component is selected.*
- virtual void **select** ()
 

*Select the component.*
- virtual void **deselect** ()
 

*Deselect the component.*
- virtual void **ToggleVisible** ()
 

*Toggle the visibility of the component.*
- virtual void **SetVisible** (bool visible)
 

*Set the visibility of the component.*
- virtual bool **GetVisible** ()
 

*Get the visibility of the component.*
- void **SetPosition** (float x, float y)
 

*Set the position of the component.*
- void **SetPosition** (Vector2 position)
 

*Set the position of the component.*
- Vector2 **GetPosition** ()
 

*Get the position of the component.*

## Static Public Attributes

- static constexpr float **mNodeDistance** = 20

## Protected Member Functions

- virtual void **UpdateMouseCursorWhenHover** (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)
 

*Update the mouse cursor when hovering.*
- virtual void **UpdateMouseCursorWhenHover** (Rectangle bound, bool hover, bool noHover)
 

*Update the mouse cursor when hovering.*
- virtual bool **GetHoverStatus** (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)
 

*Get the hover status of the component.*
- virtual bool **GetHoverStatus** (Rectangle bound, bool hover, bool noHover)
 

*Get the hover status of the component.*

## Protected Attributes

- **Core::Deque< Component::Ptr > mChildren**
- Vector2 **mPosition**

*The position of the component.*
- bool **mVisible**

*The visibility of the component.*

## Private Member Functions

- Vector2 **GetNodeDefaultPosition** (std::size\_t index)

## Private Attributes

- `FontHolder * fonts`
- `std::vector< Node > list`
- `Node::Shape mShape`
- `std::size_t capacity`
- `std::size_t length`
- `bool mIsSelected`

*The selected status of the component.*

### 7.23.1 Detailed Description

The base class for the dynamic array visualization. This class provides the basic functionality for the dynamic array visualization.

### 7.23.2 Member Typedef Documentation

#### 7.23.2.1 Ptr

```
typedef std::shared_ptr< Container > GUI::Container::Ptr [inherited]
```

The shared pointer to the container.

The `GUI` container is commonly used by multiple components, so a shared pointer is used.

See also

```
std::shared_ptr
```

### 7.23.3 Constructor & Destructor Documentation

#### 7.23.3.1 DynamicArray() [1/2]

```
GUIVisualizer::DynamicArray::DynamicArray ( )
```

The default constructor of the dynamic array visualization.

#### 7.23.3.2 DynamicArray() [2/2]

```
GUIVisualizer::DynamicArray::DynamicArray ( FontHolder * fonts )
```

The constructor of the dynamic array visualization.

**Parameters**

<i>fonts</i>	The fonts that are used to draw the text.
--------------	---

**7.23.3.3 ~DynamicArray()**

```
GUIVisualizer::DynamicArray::~DynamicArray ( )
```

The destructor of the dynamic array visualization.

**7.23.4 Member Function Documentation****7.23.4.1 Clear()**

```
void GUIVisualizer::DynamicArray::Clear ( )
```

Clear the dynamic array visualization.

**7.23.4.2 DeleteNode()**

```
void GUIVisualizer::DynamicArray::DeleteNode ( std::size_t index, bool rePosition = true )
```

Delete the node at the given index.

**Parameters**

<i>index</i>	The index to delete the node at.
<i>rePosition</i>	Whether to reposition the nodes after deleting.

**See also**

[Relayout](#)

**7.23.4.3 deselect()**

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

#### 7.23.4.4 Draw() [1/2]

```
void GUI::Container::Draw (
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Draw the container.

This function draws the container.

##### Parameters

<i>base</i>	The base position of the container.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#).

#### 7.23.4.5 Draw() [2/2]

```
void GUIVisualizer::DynamicArray::Draw (
    Vector2 base = (Vector2){0, 0},
    float t = 1.0f,
    bool init = false )
```

Draw the dynamic array visualization.

##### Parameters

<i>target</i>	The target to draw the dynamic array visualization on.
<i>states</i>	The states used to draw the dynamic array visualization.

#### 7.23.4.6 DrawCurrent()

```
void GUI::Container::DrawCurrent (
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

#### 7.23.4.7 **GenerateNode()**

```
GUIVisualizer::Node GUIVisualizer::DynamicArray::GenerateNode (
    int value )
```

#### 7.23.4.8 **GetCapacity()**

```
std::size_t GUIVisualizer::DynamicArray::GetCapacity ( ) const
```

Get the capacity of the dynamic array visualization.

##### Returns

The capacity of the dynamic array visualization.

#### 7.23.4.9 **GetCapacityFromLength()**

```
std::size_t GUIVisualizer::DynamicArray::GetCapacityFromLength (
    std::size_t length )
```

#### 7.23.4.10 **GetChildren()**

```
Core::Deque< GUI::Component::Ptr > GUI::Container::GetChildren ( ) [inherited]
```

Get the pack components of the container.

This function returns the all of the pack components of the container.

##### Returns

The children of the container.

#### 7.23.4.11 **GetHoverStatus() [1/2]**

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

**7.23.4.12 GetHoverStatus() [2/2]**

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.23.4.13 GetLength()**

```
std::size_t GUIVisualizer::DynamicArray::GetLength ( ) const
```

Get the length of the dynamic array visualization.

**Returns**

The length of the dynamic array visualization.

**7.23.4.14 GetList()**

```
std::vector< GUIVisualizer::Node > & GUIVisualizer::DynamicArray::GetList ( )
```

Get the list of nodes.

**Returns**

The list of nodes.

#### 7.23.4.15 GetNodeDefaultPosition()

```
Vector2 GUIVisualizer::DynamicArray::GetNodeDefaultPosition (
    std::size_t index ) [private]
```

#### 7.23.4.16 GetPosition()

```
Vector2 GUI::Component::GetPosition () [inherited]
```

Get the position of the component.

This function gets the position of the component.

##### Returns

The position of the component.

#### 7.23.4.17 GetShape()

```
GUIVisualizer::Node::Shape GUIVisualizer::DynamicArray::GetShape () const
```

Get the shape of the dynamic array visualization.

##### Returns

The shape of the dynamic array visualization.

#### 7.23.4.18 GetSize()

```
Vector2 GUI::Container::GetSize () [virtual], [inherited]
```

Get the size of the container.

This function returns the size of the container.

##### Returns

The size of the container.

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#), and [GUIComponent::OptionInputField](#).

#### 7.23.4.19 GetVisible()

```
bool GUI::Component::GetVisible ( ) [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

##### Returns

The visibility of the component.

#### 7.23.4.20 Import()

```
void GUIVisualizer::DynamicArray::Import ( std::vector< int > nodes )
```

Initialize the dynamic array visualization with the given nodes.

##### Parameters

<i>nodes</i>	The nodes to initialize the dynamic array visualization with.
--------------	---

#### 7.23.4.21 InsertNode()

```
void GUIVisualizer::DynamicArray::InsertNode ( std::size_t index, GUIVisualizer::Node node, bool rePosition = true )
```

Insert the given node at the given index.

##### Parameters

<i>index</i>	The index to insert the node at.
<i>node</i>	The node to insert.
<i>rePosition</i>	Whether to reposition the nodes after inserting.

##### See also

[Relayout](#)

#### 7.23.4.22 `isSelectable()`

```
bool GUIVisualizer::DynamicArray::isSelectable ( ) const [virtual]
```

Check if the container is selectable.

**Deprecated** This function is deprecated.

This function checks if the container is selectable.

##### Returns

True if the container is selectable.

Reimplemented from [GUI::Container](#).

#### 7.23.4.23 `isSelected()`

```
bool GUI::Component::isSelected ( ) const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.23.4.24 `operator[]( ) [1/2]`

```
GUIVisualizer::Node & GUIVisualizer::DynamicArray::operator[ ] (
    std::size_t index )
```

Get the node at the given index.

##### Parameters

<i>index</i>	The index of the node to get.
--------------	-------------------------------

**Returns**

The node at the given index.

**7.23.4.25 operator[]( ) [2/2]**

```
const GUIVisualizer::Node & GUIVisualizer::DynamicArray::operator[ ] ( std::size_t index ) const
```

Get the node at the given index.

**Parameters**

<i>index</i>	The index of the node to get.
--------------	-------------------------------

**Returns**

The node at the given index.

**7.23.4.26 pack()**

```
void GUI::Container::pack ( Component::Ptr component ) [inherited]
```

Pack a component into the container.

This function packs a component into the container.

**Parameters**

<i>component</i>	The component to pack.
------------------	------------------------

**7.23.4.27 Relayout()**

```
void GUIVisualizer::DynamicArray::Relayout ( )
```

Relayout the array visualization.

#### 7.23.4.28 Reserve()

```
void GUIVisualizer::DynamicArray::Reserve (
    std::size_t size )
```

Reserve the given size of the dynamic array visualization (more specific, adjust the capacity of the array).

**Parameters**

<i>size</i>	The size to reserve.
-------------	----------------------

**7.23.4.29 Resize()**

```
void GUIVisualizer::DynamicArray::Resize (
    std::size_t size )
```

Resize the dynamic array visualization.

**Parameters**

<i>size</i>	The size to resize to.
-------------	------------------------

**7.23.4.30 select()**

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

**7.23.4.31 SetPosition() [1/2]**

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

#### 7.23.4.32 SetPosition() [2/2]

```
void GUI::Component::SetPosition (
    Vector2 position )  [inherited]
```

Set the position of the component.

This function sets the position of the component.

##### Parameters

<i>position</i>	The position of the component.
-----------------	--------------------------------

#### 7.23.4.33 SetShape()

```
void GUIVisualizer::DynamicArray::SetShape (
    Node::Shape shape )
```

Set the shape of the dynamic array visualization.

##### Parameters

<i>shape</i>	The shape of the dynamic array visualization.
--------------	---

#### 7.23.4.34 SetVisible()

```
void GUI::Component::SetVisible (
    bool visible )  [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

##### Parameters

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

#### 7.23.4.35 ToggleVisible()

```
void GUI::Component::ToggleVisible ( )  [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.23.4.36 UnpackAll()

```
void GUI::Container::UnpackAll ( ) [inherited]
```

Unpack all components from the container.

This function unpacks all components from the container.

See also

[pack](#)

#### 7.23.4.37 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.23.4.38 UpdateMouseCursorWhenHover() [2/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

## 7.23.5 Member Data Documentation

### 7.23.5.1 capacity

```
std::size_t GUIVisualizer::DynamicArray::capacity [private]
```

### 7.23.5.2 fonts

```
FontHolder* GUIVisualizer::DynamicArray::fonts [private]
```

### 7.23.5.3 length

```
std::size_t GUIVisualizer::DynamicArray::length [private]
```

### 7.23.5.4 list

```
std::vector< Node > GUIVisualizer::DynamicArray::list [private]
```

### 7.23.5.5 mChildren

```
Core::Deque< Component::Ptr > GUI::Container::mChildren [protected], [inherited]
```

### 7.23.5.6 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

### 7.23.5.7 mNodeDistance

```
constexpr float GUIVisualizer::DynamicArray::mNodeDistance = 20 [static], [constexpr]
```

### 7.23.5.8 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

### 7.23.5.9 mShape

```
Node::Shape GUIVisualizer::DynamicArray::mShape [private]
```

### 7.23.5.10 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

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

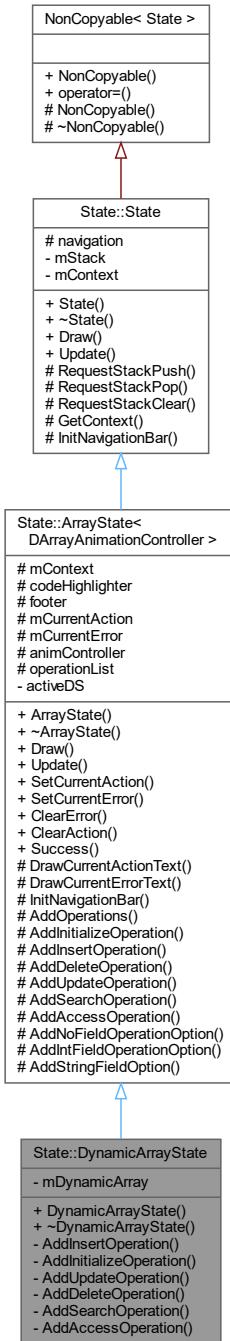
- src/Components/Visualization/[DynamicArray.hpp](#)
- src/Components/Visualization/[DynamicArray.cpp](#)

## 7.24 State::DynamicArrayState Class Reference

The state class that is used to represent the dynamic array state/scene of the application.

```
#include <DynamicArrayState.hpp>
```

Inheritance diagram for State::DynamicArrayState:



## Public Types

- `typedef std::unique_ptr< State > Ptr`

*A unique pointer to the state object.*

## Public Member Functions

- **DynamicArrayState** (`StateStack &stack, Context context`)  
*Construct a new `DynamicArrayState` object.*
- **~DynamicArrayState** ()  
*Destroy the `DynamicArrayState` object.*
- **virtual void Draw** ()  
*Draw the array state to the screen.*
- **virtual bool Update** (float dt)  
*Update the array state.*
- **virtual void SetCurrentAction** (std::string action)  
*Set the current action text.*
- **virtual void SetCurrentError** (std::string error)  
*Set the current error text.*
- **virtual void ClearError** ()  
*Clear the current error text.*
- **virtual void ClearAction** ()  
*Clear the current action text.*
- **virtual void Success** ()  
*Clear the current error and toggle the action list.*

## Protected Member Functions

- **virtual void DrawCurrentActionText** ()
- **virtual void DrawCurrentErrorText** ()
- **void InitNavigationBar** ()
- **virtual void AddOperations** ()  
*Add the operations to the operation list.*
- **virtual void AddNoFieldOperationOption** (`GUIComponent::OperationContainer::Ptr container, std::string title, std::function< void() > action`)
- **virtual void AddIntegerFieldOption** (`GUIComponent::OperationContainer::Ptr container, std::string title, Core::Deque< IntegerInput > fields, std::function< void(std::map< std::string, std::string >) > action`)
- **virtual void AddStringFieldOption** (`GUIComponent::OperationContainer::Ptr container, std::string title, std::string label, std::function< void(std::map< std::string, std::string >) > action`)
- **void RequestStackPush** (`States::ID stateID`)  
*Request the state stack to push a new state/scene.*
- **void RequestStackPop** ()  
*Request the state stack to pop the current state/scene.*
- **void RequestStackClear** ()  
*Request the state stack to clear all the states/scenes.*
- **Context GetContext** () const  
*Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).*

## Protected Attributes

- **Context mContext**
- **GUIComponent::CodeHighlighter::Ptr codeHighlighter**
- **GUIComponent::Footer< DArrayAnimationController > footer**
- **std::string mCurrentAction**
- **std::string mCurrentError**
- **T::Ptr animController**
- **GUIComponent::OperationList operationList**
- **GUIComponent::NavigationBar navigation**

## Private Member Functions

- void [AddInsertOperation \(\)](#)  
*Add an insert operation to the dynamic array algorithm.*
- void [AddInitializeOperation \(\)](#)  
*Add an initialize operation to the dynamic array algorithm.*
- void [AddUpdateOperation \(\)](#)  
*Add an update operation to the dynamic array algorithm.*
- void [AddDeleteOperation \(\)](#)  
*Add a delete operation to the dynamic array algorithm.*
- void [AddSearchOperation \(\)](#)  
*Add a search operation to the dynamic array algorithm.*
- void [AddAccessOperation \(\)](#)  
*Add an access operation to the dynamic array algorithm.*

## Private Attributes

- [Algorithm::DynamicArray mDynamicArray](#)  
*The algorithm of the dynamic array.*
- [DataStructures::ID activeDS](#)
- [StateStack \\* mStack](#)  
*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*

### 7.24.1 Detailed Description

The state class that is used to represent the dynamic array state/scene of the application.

### 7.24.2 Member Typedef Documentation

#### 7.24.2.1 Ptr

```
typedef std::unique_ptr< State > State::State::Ptr [inherited]
```

A unique pointer to the state object.

### 7.24.3 Constructor & Destructor Documentation

#### 7.24.3.1 DynamicArrayState()

```
State::DynamicArrayState::DynamicArrayState (
    StateStack & stack,
    Context context )
```

Construct a new [DynamicArrayState](#) object.

**Parameters**

<i>stack</i>	The state stack where the dynamic array state is pushed to.
<i>context</i>	The context of the application.

**7.24.3.2 ~DynamicArrayState()**

```
State::DynamicArrayState::~DynamicArrayState ( )
```

Destroy the [DynamicArrayState](#) object.

**7.24.4 Member Function Documentation****7.24.4.1 AddAccessOperation()**

```
void State::DynamicArrayState::AddAccessOperation ( ) [private], [virtual]
```

Add an access operation to the dynamic array algorithm.

Reimplemented from [State::ArrayState< DArrayAnimationController >](#).

**7.24.4.2 AddDeleteOperation()**

```
void State::DynamicArrayState::AddDeleteOperation ( ) [private], [virtual]
```

Add a delete operation to the dynamic array algorithm.

Reimplemented from [State::ArrayState< DArrayAnimationController >](#).

**7.24.4.3 AddInitializeOperation()**

```
void State::DynamicArrayState::AddInitializeOperation ( ) [private], [virtual]
```

Add an initialize operation to the dynamic array algorithm.

Reimplemented from [State::ArrayState< DArrayAnimationController >](#).

#### 7.24.4.4 AddInsertOperation()

```
void State::DynamicArrayState::AddInsertOperation ( ) [private], [virtual]
```

Add an insert operation to the dynamic array algorithm.

Reimplemented from [State::ArrayState< DArrayAnimationController >](#).

#### 7.24.4.5 AddIntFieldOperationOption()

```
void State::ArrayState< DArrayAnimationController >::AddIntFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    Core::Deque< IntegerInput > fields,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.24.4.6 AddNoFieldOperationOption()

```
void State::ArrayState< DArrayAnimationController >::AddNoFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::function< void() > action ) [protected], [virtual], [inherited]
```

#### 7.24.4.7 AddOperations()

```
void State::ArrayState< DArrayAnimationController >::AddOperations [protected], [virtual],
[inherited]
```

Add the operations to the operation list.

##### Note

This function should not be overridden as it calls the other Add\*Operation functions.

#### 7.24.4.8 AddSearchOperation()

```
void State::DynamicArrayState::AddSearchOperation ( ) [private], [virtual]
```

Add a search operation to the dynamic array algorithm.

Reimplemented from [State::ArrayState< DArrayAnimationController >](#).

#### 7.24.4.9 AddStringFieldOption()

```
void State::ArrayState< DArrayAnimationController >::AddStringFieldOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::string label,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.24.4.10 AddUpdateOperation()

```
void State::DynamicArrayState::AddUpdateOperation () [private], [virtual]
```

Add an update operation to the dynamic array algorithm.

Reimplemented from [State::ArrayState< DArrayAnimationController >](#).

#### 7.24.4.11 ClearAction()

```
void State::ArrayState< DArrayAnimationController >::ClearAction [inline], [virtual], [inherited]
```

Clear the current action text.

#### 7.24.4.12 ClearError()

```
void State::ArrayState< DArrayAnimationController >::ClearError [inline], [virtual], [inherited]
```

Clear the current error text.

#### 7.24.4.13 Draw()

```
void State::ArrayState< DArrayAnimationController >::Draw [inline], [virtual], [inherited]
```

Draw the array state to the screen.

Implements [State::State](#).

#### 7.24.4.14 DrawCurrentActionText()

```
void State::ArrayState< DArrayAnimationController >::DrawCurrentActionText [inline], [protected],  
[virtual], [inherited]
```

#### 7.24.4.15 DrawCurrentErrorText()

```
void State::ArrayState< DArrayAnimationController >::DrawCurrentErrorText [inline], [protected],  
[virtual], [inherited]
```

#### 7.24.4.16 GetContext()

```
State::State::Context State::State::GetContext () const [protected], [inherited]
```

Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### Returns

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### See also

[Context](#)

#### 7.24.4.17 InitNavigationBar()

```
void State::ArrayState< DArrayAnimationController >::InitNavigationBar [protected], [inherited]
```

#### 7.24.4.18 RequestStackClear()

```
void State::State::RequestStackClear () [protected], [inherited]
```

Request the state stack to clear all the states/scenes.

##### See also

[StateStack::ClearStates](#)

#### 7.24.4.19 RequestStackPop()

```
void State::State::RequestStackPop ( ) [protected], [inherited]
```

Request the state stack to pop the current state/scene.

See also

[StateStack::PopState](#)

#### 7.24.4.20 RequestStackPush()

```
void State::State::RequestStackPush (
    States::ID stateID ) [protected], [inherited]
```

Request the state stack to push a new state/scene.

Parameters

<i>stateID</i>	The ID of the state/scene that will be pushed
----------------	---

See also

[StateStack::PushState](#)

#### 7.24.4.21 SetCurrentAction()

```
void State::ArrayState< DArrayAnimationController >::SetCurrentAction (
    std::string action ) [inline], [virtual], [inherited]
```

Set the current action text.

Parameters

<i>action</i>	The current action text.
---------------	--------------------------

#### 7.24.4.22 SetCurrentError()

```
void State::ArrayState< DArrayAnimationController >::SetCurrentError (
    std::string error ) [inline], [virtual], [inherited]
```

Set the current error text.

**Parameters**

<code>error</code>	The current error text.
--------------------	-------------------------

**7.24.4.23 Success()**

```
void State::ArrayState< DArrayAnimationController >::Success [inline], [virtual], [inherited]
```

Clear the current error and toggle the action list.

**7.24.4.24 Update()**

```
bool State::ArrayState< DArrayAnimationController >::Update (
    float dt ) [virtual], [inherited]
```

Update the array state.

**Parameters**

<code>dt</code>	The delta time between the previous and the current frame.
-----------------	--

**Returns**

`true` If the update is successful.  
`false` If the update is unsuccessful.

Implements [State::State](#).

**7.24.5 Member Data Documentation****7.24.5.1 activeDS**

```
DataStructures::ID State::ArrayState< DArrayAnimationController >::activeDS [private], [inherited]
```

**7.24.5.2 animController**

```
T::Ptr State::ArrayState< DArrayAnimationController >::animController [protected], [inherited]
```

#### 7.24.5.3 codeHighlighter

```
GUIComponent::CodeHighlighter::Ptr State::ArrayState< DArrayAnimationController >::codeHighlighter [protected], [inherited]
```

#### 7.24.5.4 footer

```
GUIComponent::Footer< DArrayAnimationController > State::ArrayState< DArrayAnimationController >::footer [protected], [inherited]
```

#### 7.24.5.5 mContext

```
Context State::ArrayState< DArrayAnimationController >::mContext [protected], [inherited]
```

#### 7.24.5.6 mCurrentAction

```
std::string State::ArrayState< DArrayAnimationController >::mCurrentAction [protected], [inherited]
```

#### 7.24.5.7 mCurrentError

```
std::string State::ArrayState< DArrayAnimationController >::mCurrentError [protected], [inherited]
```

#### 7.24.5.8 mDynamicArray

```
Algorithm::DynamicArray State::DynamicArrayState::mDynamicArray [private]
```

The algorithm of the dynamic array.

#### 7.24.5.9 mStack

```
StateStack* State::State::mStack [private], [inherited]
```

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

#### 7.24.5.10 navigation

```
GUIComponent::NavigationBar State::State::navigation [protected], [inherited]
```

#### 7.24.5.11 operationList

```
GUIComponent::OperationList State::ArrayState< DArrayAnimationController >::operationList  
[protected], [inherited]
```

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

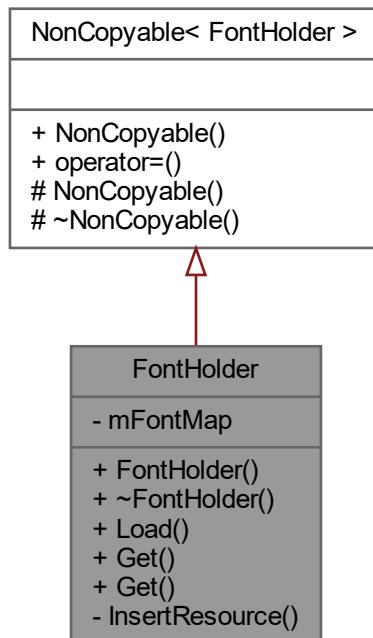
- src/States/Array/[DynamicArrayList.hpp](#)
- src/States/Array/[DynamicArrayList.cpp](#)

## 7.25 FontHolder Class Reference

The font holder class that is used to store the fonts.

```
#include <FontHolder.hpp>
```

Inheritance diagram for FontHolder:



## Public Member Functions

- [FontHolder \(\)](#)  
*The constructor.*
- [~FontHolder \(\)](#)  
*The destructor.*
- [void Load \(Fonts::ID id, const std::string &filename\)](#)  
*Load the font from the file.*
- [Font & Get \(Fonts::ID id\)](#)  
*Retrieve the font.*
- [const Font & Get \(Fonts::ID id\) const](#)  
*Retrieve the font.*

## Private Member Functions

- [void InsertResource \(Fonts::ID id, std::unique\\_ptr<Font> font\)](#)  
*Insert the font.*

## Private Attributes

- [std::map<Fonts::ID, std::unique\\_ptr<Font>> mFontMap](#)  
*The font map.*

### 7.25.1 Detailed Description

The font holder class that is used to store the fonts.

The font holder class is used to store the fonts. Also, this class is using the flyweight pattern to store the fonts.

#### See also

[NonCopyable](#)

[Font](#)

### 7.25.2 Constructor & Destructor Documentation

#### 7.25.2.1 [FontHolder\(\)](#)

```
FontHolder::FontHolder ( )
```

The constructor.

The constructor that is used to initialize the font holder.

### 7.25.2.2 ~FontHolder()

```
FontHolder::~FontHolder ( )
```

The destructor.

The destructor that is used to destroy the font holder.

## 7.25.3 Member Function Documentation

### 7.25.3.1 Get() [1/2]

```
Font & FontHolder::Get (
    Fonts::ID id )
```

Retrieve the font.

This function is used to retrieve the font.

#### Parameters

<i>id</i>	The font ID.
-----------	--------------

#### Returns

The font.

#### See also

Font::ID

### 7.25.3.2 Get() [2/2]

```
const Font & FontHolder::Get (
    Fonts::ID id ) const
```

Retrieve the font.

This function is used to retrieve the font.

#### Parameters

<i>id</i>	The font ID.
-----------	--------------

**Returns**

The font.

**See also**

[Font::ID](#)

### 7.25.3.3 InsertResource()

```
void FontHolder::InsertResource (
    Fonts::ID id,
    std::unique_ptr<Font> font ) [private]
```

Insert the font.

This function is used to insert the font.

**Parameters**

<i>id</i>	The font ID.
<i>font</i>	The font.

**See also**

[Font::ID](#)

### 7.25.3.4 Load()

```
void FontHolder::Load (
    Fonts::ID id,
    const std::string & filename )
```

Load the font from the file.

This function is used to load the font from the file.

**Parameters**

<i>id</i>	The font ID.
<i>filename</i>	The font filename.

**See also**

[Font::ID](#)

## 7.25.4 Member Data Documentation

### 7.25.4.1 mFontMap

```
std::map< Fonts::ID, std::unique_ptr< Font > > FontHolder::mFontMap [private]
```

The font map.

The font map that is used to store the fonts. As the flyweight pattern is used, the fonts are stored as the pointers and should be stored in the map.

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

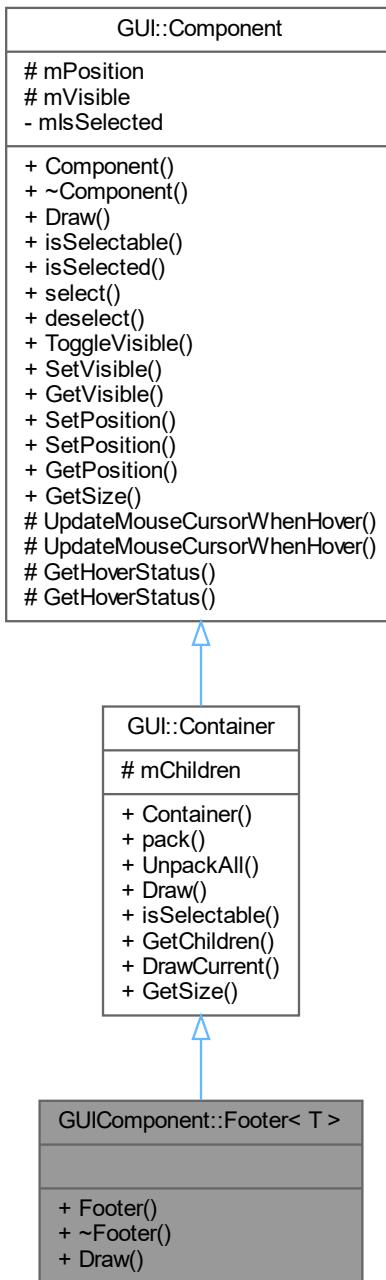
- src/[FontHolder.hpp](#)
- src/[FontHolder.cpp](#)

## 7.26 GUIComponent::Footer< T > Class Template Reference

The footer class that is used to represent a footer in the [GUI](#).

```
#include <Footer.hpp>
```

Inheritance diagram for GUIComponent::Footer< T >:



## Public Types

- `typedef std::shared_ptr< Container > Ptr`

*The shared pointer to the container.*

## Public Member Functions

- `Footer ()`
- `~Footer ()`
- `void Draw (T *animController, Vector2 base=(Vector2){0, 0})`
- `void pack (Component::Ptr component)`  
`Pack a component into the container.`
- `void UnpackAll ()`  
`Unpack all components from the container.`
- `virtual void Draw (Vector2 base=(Vector2){0, 0})`  
`Draw the container.`
- `virtual bool IsSelectable () const`  
`Check if the container is selectable.`
- `Core::Deque< Component::Ptr > GetChildren ()`  
`Get the pack components of the container.`
- `virtual void DrawCurrent (Vector2 base)`
- `virtual Vector2 GetSize ()`  
`Get the size of the container.`
- `bool IsSelected () const`  
`Check if the component is selected.`
- `virtual void select ()`  
`Select the component.`
- `virtual void deselect ()`  
`Deselect the component.`
- `virtual void ToggleVisible ()`  
`Toggle the visibility of the component.`
- `virtual void SetVisible (bool visible)`  
`Set the visibility of the component.`
- `virtual bool GetVisible ()`  
`Get the visibility of the component.`
- `void SetPosition (float x, float y)`  
`Set the position of the component.`
- `void SetPosition (Vector2 position)`  
`Set the position of the component.`
- `Vector2 GetPosition ()`  
`Get the position of the component.`

## Protected Member Functions

- `virtual void UpdateMouseCursorWhenHover (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
`Update the mouse cursor when hovering.`
- `virtual void UpdateMouseCursorWhenHover (Rectangle bound, bool hover, bool noHover)`  
`Update the mouse cursor when hovering.`
- `virtual bool GetHoverStatus (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
`Get the hover status of the component.`
- `virtual bool GetHoverStatus (Rectangle bound, bool hover, bool noHover)`  
`Get the hover status of the component.`

## Protected Attributes

- `Core::Deque< Component::Ptr > mChildren`
- `Vector2 mPosition`  
*The position of the component.*
- `bool mVisible`  
*The visibility of the component.*

## Private Attributes

- `bool mIsSelected`  
*The selected status of the component.*

### 7.26.1 Detailed Description

```
template<typename T>
class GUIComponent::Footer< T >
```

The footer class that is used to represent a footer in the [GUI](#).

The footer is used to show the animation controller

### 7.26.2 Member Typedef Documentation

#### 7.26.2.1 Ptr

```
typedef std::shared_ptr< Container > GUI::Container::Ptr [inherited]
```

The shared pointer to the container.

The [GUI](#) container is commonly used by multiple components, so a shared pointer is used.

See also

`std::shared_ptr`

### 7.26.3 Constructor & Destructor Documentation

#### 7.26.3.1 Footer()

```
template<typename T >
GUIComponent::Footer< T >::Footer
```

### 7.26.3.2 ~Footer()

```
template<typename T >
GUIComponent::Footer< T >::~Footer
```

## 7.26.4 Member Function Documentation

### 7.26.4.1 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

### 7.26.4.2 Draw() [1/2]

```
template<typename T >
void GUIComponent::Footer< T >::Draw (
    T * animController,
    Vector2 base = (Vector2){0, 0} )
```

### 7.26.4.3 Draw() [2/2]

```
void GUI::Container::Draw (
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Draw the container.

This function draws the container.

#### Parameters

<i>base</i>	The base position of the container.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#).

#### 7.26.4.4 DrawCurrent()

```
void GUI::Container::DrawCurrent (
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

#### 7.26.4.5 GetChildren()

```
Core::Deque< GUI::Component::Ptr > GUI::Container::GetChildren () [inherited]
```

Get the pack components of the container.

This function returns the all of the pack components of the container.

##### Returns

The children of the container.

#### 7.26.4.6 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.26.4.7 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.26.4.8 GetPosition()**

```
Vector2 GUI::Component::GetPosition () [inherited]
```

Get the position of the component.

This function gets the position of the component.

**Returns**

The position of the component.

**7.26.4.9 GetSize()**

```
Vector2 GUI::Container::GetSize () [virtual], [inherited]
```

Get the size of the container.

This function returns the size of the container.

**Returns**

The size of the container.

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#), and [GUIComponent::OptionInputField](#).

**7.26.4.10 GetVisible()**

```
bool GUI::Component::GetVisible () [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

**Returns**

The visibility of the component.

#### 7.26.4.11 isSelectable()

```
bool GUI::Container::isSelectable () const [virtual], [inherited]
```

Check if the container is selectable.

**Deprecated** This function is deprecated.

This function checks if the container is selectable.

##### Returns

True if the container is selectable.

Implements [GUI::Component](#).

Reimplemented in [GUIVisualizer::CircularLinkedList](#), [GUIVisualizer::DoublyLinkedList](#), [GUIVisualizer::DynamicArray](#), [GUIVisualizer::LinkedList](#), and [GUIVisualizer::SinglyLinkedList](#).

#### 7.26.4.12 isSelected()

```
bool GUI::Component::isSelected () const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.26.4.13 pack()

```
void GUI::Container::pack (
    Component::Ptr component ) [inherited]
```

Pack a component into the container.

This function packs a component into the container.

**Parameters**

<i>component</i>	The component to pack.
------------------	------------------------

**7.26.4.14 select()**

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

**7.26.4.15 SetPosition() [1/2]**

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

**7.26.4.16 SetPosition() [2/2]**

```
void GUI::Component::SetPosition (
    Vector2 position ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>position</i>	The position of the component.
-----------------	--------------------------------

#### 7.26.4.17 SetVisible()

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

##### Parameters

<code>visible</code>	The visibility of the component.
----------------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

#### 7.26.4.18 ToggleVisible()

```
void GUI::Component::ToggleVisible () [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.26.4.19 UnpackAll()

```
void GUI::Container::UnpackAll () [inherited]
```

Unpack all components from the container.

This function unpacks all components from the container.

##### See also

[pack](#)

#### 7.26.4.20 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

**7.26.4.21 UpdateMouseCursorWhenHover() [2/2]**

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.26.5 Member Data Documentation****7.26.5.1 mChildren**

```
Core::Deque< Component::Ptr > GUI::Container::mChildren [protected], [inherited]
```

**7.26.5.2 mIsSelected**

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

**7.26.5.3 mPosition**

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

#### 7.26.5.4 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

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

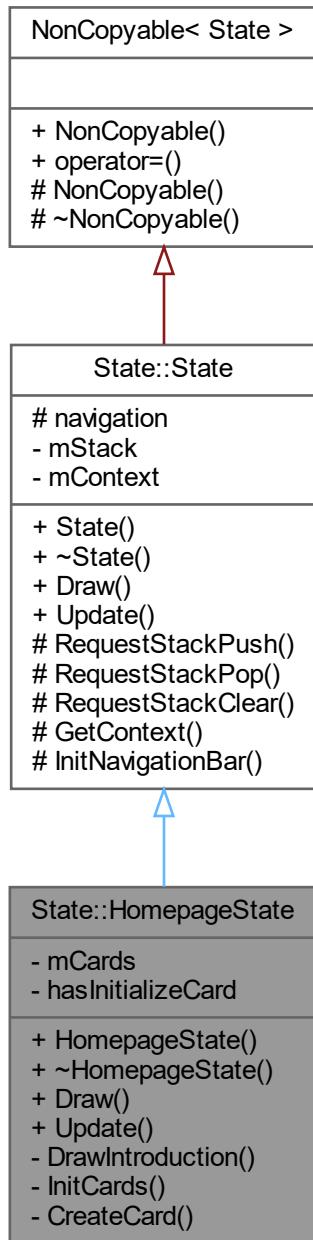
- src/Components/Common/[Footer.hpp](#)

## 7.27 State::HomepageState Class Reference

The homepage screen.

```
#include <HomepageState.hpp>
```

Inheritance diagram for State::HomepageState:



## Public Types

- `typedef std::unique_ptr< State > Ptr`  
*A unique pointer to the state object.*

## Public Member Functions

- `HomepageState (StateStack &stack, Context context)`

- `Construct a new HomepageState object.`
- `~HomepageState ()`  
*Destroy the [HomepageState](#) scene.*
- `void Draw ()`  
*Draw the homepage scene.*
- `bool Update (float dt)`  
*Update the components of the homepage scene.*

## Protected Member Functions

- `void RequestStackPush (States::ID stateID)`  
*Request the state stack to push a new state/scene.*
- `void RequestStackPop ()`  
*Request the state stack to pop the current state/scene.*
- `void RequestStackClear ()`  
*Request the state stack to clear all the states/scenes.*
- `Context GetContext () const`  
*Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).*
- `void InitNavigationBar ()`  
*Initialize the navigation bar as a component of the state/scene.*

## Protected Attributes

- `GUIComponent::NavigationBar navigation`

## Private Member Functions

- `void DrawIntroduction ()`
- `void InitCards ()`
- `void CreateCard (States::ID stateID, std::string title, Textures::ID textureID, int x, int y)`

## Private Attributes

- `GUI::Container mCards`
- `bool hasInitializeCard`
- `StateStack * mStack`  
*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*
- `Context mContext`  
*The context that is used to share the resources (fonts, textures, ...) between states (scenes).*

### 7.27.1 Detailed Description

The homepage screen.

This class is used to display the homepage screen of the application.

## 7.27.2 Member Typedef Documentation

### 7.27.2.1 Ptr

```
typedef std::unique_ptr< State > State::State::Ptr [inherited]
```

A unique pointer to the state object.

## 7.27.3 Constructor & Destructor Documentation

### 7.27.3.1 HomepageState()

```
State::HomepageState::HomepageState (
    StateStack & stack,
    Context context )
```

Construct a new [HomepageState](#) object.

#### Parameters

<code>stack</code>	The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).
<code>context</code>	The context that is used to share the resources (fonts, textures, ...) between states (scenes).

### 7.27.3.2 ~HomepageState()

```
State::HomepageState::~HomepageState ( )
```

Destroy the [HomepageState](#) scene.

## 7.27.4 Member Function Documentation

### 7.27.4.1 CreateCard()

```
void State::HomepageState::CreateCard (
    States::ID stateID,
    std::string title,
    Textures::ID textureID,
    int x,
    int y ) [private]
```

#### 7.27.4.2 Draw()

```
void State::HomepageState::Draw ( ) [virtual]
```

Draw the homepage scene.

Implements [State::State](#).

#### 7.27.4.3 DrawIntroduction()

```
void State::HomepageState::DrawIntroduction ( ) [private]
```

#### 7.27.4.4 GetContext()

```
State::State::Context State::State::GetContext ( ) const [protected], [inherited]
```

Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### Returns

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### See also

[Context](#)

#### 7.27.4.5 InitCards()

```
void State::HomepageState::InitCards ( ) [private]
```

#### 7.27.4.6 InitNavigationBar()

```
void State::State::InitNavigationBar ( ) [protected], [inherited]
```

Initialize the navigation bar as a component of the state/scene.

##### See also

[GUIComponent::NavigationBar](#)

#### 7.27.4.7 RequestStackClear()

```
void State::State::RequestStackClear ( ) [protected], [inherited]
```

Request the state stack to clear all the states/scenes.

See also

[StateStack::ClearStates](#)

#### 7.27.4.8 RequestStackPop()

```
void State::State::RequestStackPop ( ) [protected], [inherited]
```

Request the state stack to pop the current state/scene.

See also

[StateStack::PopState](#)

#### 7.27.4.9 RequestStackPush()

```
void State::State::RequestStackPush ( States::ID stateID ) [protected], [inherited]
```

Request the state stack to push a new state/scene.

Parameters

<code>stateID</code>	The ID of the state/scene that will be pushed
----------------------	---

See also

[StateStack::PushState](#)

#### 7.27.4.10 Update()

```
bool State::HomepageState::Update ( float dt ) [virtual]
```

Update the the components of the homepage scene.

**Parameters**

<i>dt</i>	The time interval between the previous and the new frame
-----------	--

**Returns**

true (always)

Implements [State::State](#).

## 7.27.5 Member Data Documentation

### 7.27.5.1 hasInitializeCard

```
bool State::HomepageState::hasInitializeCard [private]
```

### 7.27.5.2 mCards

```
GUI::Container State::HomepageState::mCards [private]
```

### 7.27.5.3 mContext

```
Context State::State::mContext [private], [inherited]
```

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

### 7.27.5.4 mStack

```
StateStack* State::State::mStack [private], [inherited]
```

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

### 7.27.5.5 navigation

```
GUIComponent::NavigationBar State::State::navigation [protected], [inherited]
```

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

- src/States/HomepageState.hpp
- src/States/HomepageState.cpp

## 7.28 CategoryInfo::Info Struct Reference

The info struct that is used to store information about the categories.

### Public Member Functions

- [Info \(Category::ID categoryID, std::vector< DataStructures::ID > mDS, std::string name\)](#)  
*The constructor.*

### Public Attributes

- [Category::ID categoryID](#)  
*The category ID.*
- [std::string categoryName](#)  
*The category name.*
- [std::vector< DataStructures::ID > mDS](#)  
*The data structures that are in the category.*

### 7.28.1 Detailed Description

The info struct that is used to store information about the categories.

The info struct is used to store information about the categories. This struct is used to store the category ID, the category name, and the data structures that are in the category.

### 7.28.2 Constructor & Destructor Documentation

#### 7.28.2.1 Info()

```
CategoryInfo::Info::Info (
    Category::ID categoryID,
    std::vector< DataStructures::ID > mDS,
    std::string name )
```

The constructor.

The constructor that is used to initialize the category info struct.

**Parameters**

<i>categoryID</i>	The category ID.
<i>mDS</i>	The data structures that are in the category.
<i>name</i>	The category name.

## 7.28.3 Member Data Documentation

### 7.28.3.1 categoryID

[Category::ID](#) `CategoryInfo::Info::categoryID`

The category ID.

**See also**

[Category::ID](#)

### 7.28.3.2 categoryName

`std::string CategoryInfo::Info::categoryName`

The category name.

### 7.28.3.3 mDS

`std::vector< DataStructures::ID > CategoryInfo::Info::mDS`

The data structures that are in the category.

**See also**

[DataStructures::ID](#)

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

- src/Identifiers/[CategoryInfo.hpp](#)
- src/Identifiers/[CategoryInfo.cpp](#)

## 7.29 DSInfo::Info Struct Reference

The info struct that is used to store information about the data structures.

### Public Member Functions

- `Info (DataStructures::ID ID, States::ID stateID, Category::ID categoryID, Textures::ID thumbnail, std::string name, std::string abbr)`

*The constructor.*

### Public Attributes

- `DataStructures::ID ID`

*The data structure ID.*

- `States::ID stateID`

*The state ID.*

- `Category::ID categoryID`

*The category ID.*

- `Textures::ID thumbnail`

*The thumbnail texture ID.*

- `std::string name`

*The data structure name.*

- `std::string abbr`

*The data structure abbreviation.*

### 7.29.1 Detailed Description

The info struct that is used to store information about the data structures.

The info struct is used to store information about the data structures. This struct is used to store the data structure ID, the state ID, the category ID, the thumbnail texture ID, the data structure name, and the data structure abbreviation.

### 7.29.2 Constructor & Destructor Documentation

#### 7.29.2.1 Info()

```
DSInfo::Info::Info (
    DataStructures::ID ID,
    States::ID stateID,
    Category::ID categoryID,
    Textures::ID thumbnail,
    std::string name,
    std::string abbr )
```

The constructor.

The constructor that is used to initialize the data structure info struct.

**Parameters**

<i>ID</i>	The data structure ID.
<i>stateID</i>	The state ID.
<i>categoryID</i>	The category ID.
<i>thumbnail</i>	The thumbnail texture ID.
<i>name</i>	The data structure name.
<i>abbr</i>	The data structure abbreviation.

## 7.29.3 Member Data Documentation

### 7.29.3.1 abbr

`std::string DSInfo::Info::abbr`

The data structure abbreviation.

### 7.29.3.2 categoryID

`Category::ID DSInfo::Info::categoryID`

The category ID.

**See also**

[Category::ID](#)

### 7.29.3.3 ID

`DataStructures::ID DSInfo::Info::ID`

The data structure ID.

**See also**

[DataStructures::ID](#)

#### 7.29.3.4 name

`std::string DSInfo::Info::name`

The data structure name.

#### 7.29.3.5 stateID

`States::ID DSInfo::Info::stateID`

The state ID.

**See also**

[States::ID](#)

#### 7.29.3.6 thumbnail

`Textures::ID DSInfo::Info::thumbnail`

The thumbnail texture ID.

**See also**

[Textures::ID](#)

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

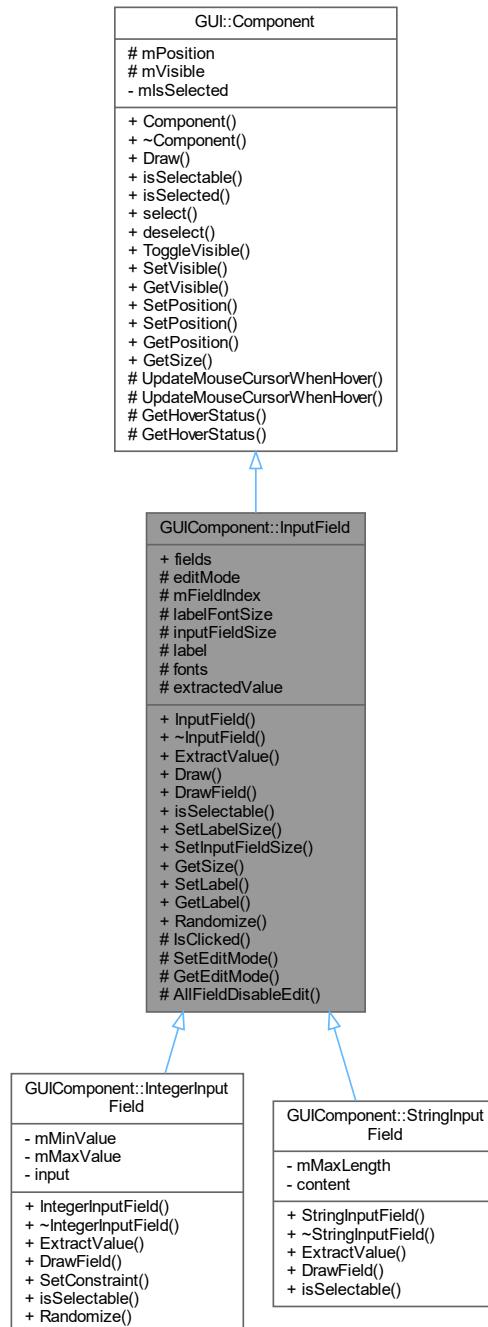
- src/Identifiers/[DSInfo.hpp](#)
- src/Identifiers/[DSInfo.cpp](#)

## 7.30 GUIComponent::InputField Class Reference

The input field class that is used to represent an input field in the [GUI](#).

```
#include <InputField.hpp>
```

Inheritance diagram for GUIComponent::InputField:



### Public Types

- `typedef std::shared_ptr< InputField > Ptr`

## Public Member Functions

- `InputField (FontHolder *fonts)`
- `~InputField ()`
- virtual std::string `ExtractValue ()=0`
- virtual void `Draw (Vector2 base=(Vector2){0, 0})`  
`Draw the component.`
- virtual void `DrawField (Vector2 base=(Vector2){0, 0})=0`
- virtual bool `IsSelectable () const`  
`Check if the component is selectable.`
- virtual void `SetLabelSize (float fontSize)`
- virtual void `SetInputFieldSize (Vector2 size)`
- virtual Vector2 `GetSize ()`  
`Get the size of the component.`
- virtual void `SetLabel (std::string labelContent)`
- virtual std::string `GetLabel () const`
- virtual void `Randomize ()`
- bool `isSelected () const`  
`Check if the component is selected.`
- virtual void `select ()`  
`Select the component.`
- virtual void `deselect ()`  
`Deselect the component.`
- virtual void `ToggleVisible ()`  
`Toggle the visibility of the component.`
- virtual void `SetVisible (bool visible)`  
`Set the visibility of the component.`
- virtual bool `GetVisible ()`  
`Get the visibility of the component.`
- void `SetPosition (float x, float y)`  
`Set the position of the component.`
- void `SetPosition (Vector2 position)`  
`Set the position of the component.`
- Vector2 `GetPosition ()`  
`Get the position of the component.`

## Static Public Attributes

- static std::vector< bool > `fields`

## Protected Member Functions

- virtual bool `IsClicked (Vector2 base=(Vector2){0, 0}) const`
- virtual void `SetEditMode (bool canEdit)`
- virtual bool `GetEditMode () const`
- virtual void `AllFieldDisableEdit ()`
- virtual void `UpdateMouseCursorWhenHover (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
`Update the mouse cursor when hovering.`
- virtual void `UpdateMouseCursorWhenHover (Rectangle bound, bool hover, bool noHover)`  
`Update the mouse cursor when hovering.`
- virtual bool `GetHoverStatus (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
`Get the hover status of the component.`
- virtual bool `GetHoverStatus (Rectangle bound, bool hover, bool noHover)`  
`Get the hover status of the component.`

## Protected Attributes

- bool `editMode`
- std::size\_t `mFieldIndex`
- float `labelFontSize`
- Vector2 `inputFieldSize`
- std::string `label`
- `FontHolder` \* `fonts`
- std::string `extractedValue`
- Vector2 `mPosition`  
*The position of the component.*
- bool `mVisible`  
*The visibility of the component.*

## Private Attributes

- bool `mIsSelected`  
*The selected status of the component.*

### 7.30.1 Detailed Description

The input field class that is used to represent an input field in the [GUI](#).

The input field is used to get input from the user.

### 7.30.2 Member Typedef Documentation

#### 7.30.2.1 Ptr

```
typedef std::shared_ptr< InputField > GUIComponent::InputField::Ptr
```

### 7.30.3 Constructor & Destructor Documentation

#### 7.30.3.1 InputField()

```
GUIComponent::InputField::InputField (
    FontHolder * fonts )
```

### 7.30.3.2 ~InputField()

```
GUIComponent::InputField::~InputField ( )
```

## 7.30.4 Member Function Documentation

### 7.30.4.1 AllFieldDisableEdit()

```
void GUIComponent::InputField::AllFieldDisableEdit ( ) [protected], [virtual]
```

### 7.30.4.2 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

### 7.30.4.3 Draw()

```
void GUIComponent::InputField::Draw ( 
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Draw the component.

This function draws the component.

#### Parameters

<i>base</i>	The base position of the component.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

### 7.30.4.4 DrawField()

```
virtual void GUIComponent::InputField::DrawField ( 
    Vector2 base = (Vector2){0, 0} ) [pure virtual]
```

Implemented in [GUIComponent::IntegerInputField](#), and [GUIComponent::StringInputField](#).

#### 7.30.4.5 ExtractValue()

```
virtual std::string GUIComponent::InputField::ExtractValue () [pure virtual]
```

Implemented in [GUIComponent::IntegerInputField](#), and [GUIComponent::StringInputField](#).

#### 7.30.4.6 GetEditMode()

```
bool GUIComponent::InputField::GetEditMode () const [protected], [virtual]
```

#### 7.30.4.7 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.30.4.8 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

#### 7.30.4.9 GetLabel()

```
std::string GUIComponent::InputField::GetLabel() const [virtual]
```

#### 7.30.4.10GetPosition()

```
Vector2 GUI::Component::GetPosition() [inherited]
```

Get the position of the component.

This function gets the position of the component.

##### Returns

The position of the component.

#### 7.30.4.11GetSize()

```
Vector2 GUIComponent::InputField::GetSize() [virtual]
```

Get the size of the component.

This function gets the size of the component.

##### Returns

The size of the component.

Reimplemented from [GUI::Component](#).

#### 7.30.4.12GetVisible()

```
bool GUI::Component::GetVisible() [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

##### Returns

The visibility of the component.

#### 7.30.4.13 IsClicked()

```
bool GUIComponent::InputField::IsClicked (
    Vector2 base = (Vector2){0, 0} ) const [protected], [virtual]
```

#### 7.30.4.14 IsSelectable()

```
bool GUIComponent::InputField::IsSelectable () const [virtual]
```

Check if the component is selectable.

**Deprecated** This function is deprecated.

This function checks if the component is selectable.

##### Returns

True if the component is selectable.

Implements [GUI::Component](#).

Reimplemented in [GUIComponent::IntegerInputField](#), and [GUIComponent::StringInputField](#).

#### 7.30.4.15 IsSelected()

```
bool GUI::Component::IsSelected () const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.30.4.16 Randomize()

```
void GUIComponent::InputField::Randomize () [virtual]
```

Reimplemented in [GUIComponent::IntegerInputField](#).

#### 7.30.4.17 **select()**

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

#### 7.30.4.18 **SetEditMode()**

```
void GUIComponent::InputField::SetEditMode (
    bool canEdit ) [protected], [virtual]
```

#### 7.30.4.19 **SetInputFieldSize()**

```
void GUIComponent::InputField::SetInputFieldSize (
    Vector2 size ) [virtual]
```

#### 7.30.4.20 **SetLabel()**

```
void GUIComponent::InputField::SetLabel (
    std::string labelText ) [virtual]
```

#### 7.30.4.21 **SetLabelSize()**

```
void GUIComponent::InputField::SetLabelSize (
    float fontSize ) [virtual]
```

#### 7.30.4.22 **SetPosition() [1/2]**

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

**7.30.4.23 SetPosition() [2/2]**

```
void GUI::Component::SetPosition (
    Vector2 position ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>position</i>	The position of the component.
-----------------	--------------------------------

**7.30.4.24 SetVisible()**

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

**Parameters**

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

**7.30.4.25 ToggleVisible()**

```
void GUI::Component::ToggleVisible ( ) [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.30.4.26 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.30.4.27 UpdateMouseCursorWhenHover() [2/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

### 7.30.5 Member Data Documentation

#### 7.30.5.1 editMode

```
bool GUIComponent::InputField::editMode [protected]
```

#### 7.30.5.2 extractedValue

```
std::string GUIComponent::InputField::extractedValue [protected]
```

### 7.30.5.3 fields

```
std::vector< bool > GUIComponent::InputField::fields [static]
```

### 7.30.5.4 fonts

```
FontHolder* GUIComponent::InputField::fonts [protected]
```

### 7.30.5.5 inputFieldSize

```
Vector2 GUIComponent::InputField::inputFieldSize [protected]
```

### 7.30.5.6 label

```
std::string GUIComponent::InputField::label [protected]
```

### 7.30.5.7 labelFontSize

```
float GUIComponent::InputField::labelFontSize [protected]
```

### 7.30.5.8 mFieldIndex

```
std::size_t GUIComponent::InputField::mFieldIndex [protected]
```

### 7.30.5.9 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

### 7.30.5.10 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

### 7.30.5.11 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

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

- src/Components/Common/[InputField.hpp](#)
- src/[Application.cpp](#)
- src/Components/Common/[InputField.cpp](#)

## 7.31 State::ArrayState< T >::IntegerInput Struct Reference

```
#include <ArrayType.hpp>
```

### Public Attributes

- std::string [label](#)
- int [width](#)
- int [minValue](#)
- int [maxValue](#)

### 7.31.1 Member Data Documentation

#### 7.31.1.1 [label](#)

```
template<typename T >
std::string State::ArrayType< T >::IntegerInput::label
```

### 7.31.1.2 maxValue

```
template<typename T >
int State::ArrayState< T >::IntegerInput::maxValue
```

### 7.31.1.3 minValue

```
template<typename T >
int State::ArrayState< T >::IntegerInput::minValue
```

### 7.31.1.4 width

```
template<typename T >
int State::ArrayState< T >::IntegerInput::width
```

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

- src/States/Array/ArrayState.hpp

## 7.32 State::LLState< T >::IntegerInput Struct Reference

```
#include <LLState.hpp>
```

### Public Attributes

- std::string [label](#)
- int [width](#)
- int [minValue](#)
- int [maxValue](#)

### 7.32.1 Member Data Documentation

#### 7.32.1.1 label

```
template<typename T >
std::string State::LLState< T >::IntegerInput::label
```

### 7.32.1.2 maxValue

```
template<typename T >
int State::LLState< T >::IntegerInput::maxValue
```

### 7.32.1.3 minValue

```
template<typename T >
int State::LLState< T >::IntegerInput::minValue
```

### 7.32.1.4 width

```
template<typename T >
int State::LLState< T >::IntegerInput::width
```

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

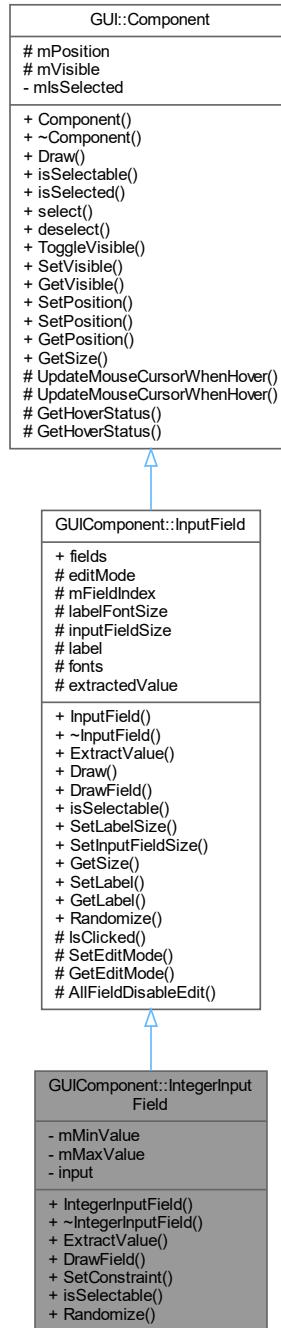
- src/States/LinkedList/[LLState.hpp](#)

## 7.33 GUIComponent::IntegerField Class Reference

The integer input field class that is used to represent an integer input field in the [GUI](#).

```
#include <IntegerField.hpp>
```

Inheritance diagram for GUIComponent::IntegerField:



## Public Types

- `typedef std::shared_ptr< IntegerIntegerField > Ptr`

## Public Member Functions

- `IntegerField (FontHolder *fonts)`

- `~IntegerField ()`
- `std::string ExtractValue ()`
- `void DrawField (Vector2 base=(Vector2){0, 0})`
- `void SetConstraint (int minValue, int maxValue)`
- `bool IsSelectable () const`  
`Check if the component is selectable.`
- `void Randomize ()`
- `virtual void Draw (Vector2 base=(Vector2){0, 0})`  
`Draw the component.`
- `virtual void SetLabelSize (float fontSize)`
- `virtual void SetInputFieldSize (Vector2 size)`
- `virtual Vector2 GetSize ()`  
`Get the size of the component.`
- `virtual void SetLabel (std::string labelContent)`
- `virtual std::string GetLabel () const`
- `bool IsSelected () const`  
`Check if the component is selected.`
- `virtual void select ()`  
`Select the component.`
- `virtual void deselect ()`  
`Deselect the component.`
- `virtual void ToggleVisible ()`  
`Toggle the visibility of the component.`
- `virtual void SetVisible (bool visible)`  
`Set the visibility of the component.`
- `virtual bool GetVisible ()`  
`Get the visibility of the component.`
- `void SetPosition (float x, float y)`  
`Set the position of the component.`
- `void SetPosition (Vector2 position)`  
`Set the position of the component.`
- `Vector2 GetPosition ()`  
`Get the position of the component.`

## Static Public Attributes

- `static std::vector< bool > fields`

## Protected Member Functions

- `virtual bool IsClicked (Vector2 base=(Vector2){0, 0}) const`
- `virtual void SetEditMode (bool canEdit)`
- `virtual bool GetEditMode () const`
- `virtual void AllFieldDisableEdit ()`
- `virtual void UpdateMouseCursorWhenHover (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
`Update the mouse cursor when hovering.`
- `virtual void UpdateMouseCursorWhenHover (Rectangle bound, bool hover, bool noHover)`  
`Update the mouse cursor when hovering.`
- `virtual bool GetHoverStatus (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
`Get the hover status of the component.`
- `virtual bool GetHoverStatus (Rectangle bound, bool hover, bool noHover)`  
`Get the hover status of the component.`

## Protected Attributes

- bool `editMode`
- std::size\_t `mFieldIndex`
- float `labelFontSize`
- Vector2 `inputFontSize`
- std::string `label`
- `FontHolder` \* `fonts`
- std::string `extractedValue`
- Vector2 `mPosition`  
*The position of the component.*
- bool `mVisible`  
*The visibility of the component.*

## Private Attributes

- int `mMinValue`
- int `mMaxValue`
- int `input`
- bool `mIsSelected`  
*The selected status of the component.*

### 7.33.1 Detailed Description

The integer input field class that is used to represent an integer input field in the [GUI](#).

The integer input field is used to get integer input from the user.

### 7.33.2 Member Typedef Documentation

#### 7.33.2.1 Ptr

```
typedef std::shared_ptr< IntegerField > GUIComponent::IntegerField::Ptr
```

### 7.33.3 Constructor & Destructor Documentation

#### 7.33.3.1 IntegerInputField()

```
GUIComponent::IntegerField::IntegerField (  
    FontHolder * fonts )
```

### 7.33.3.2 ~IntegerInputField()

```
GUIComponent::IntegerInputField::~IntegerInputField ( )
```

## 7.33.4 Member Function Documentation

### 7.33.4.1 AllFieldDisableEdit()

```
void GUIComponent::InputField::AllFieldDisableEdit ( ) [protected], [virtual], [inherited]
```

### 7.33.4.2 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

### 7.33.4.3 Draw()

```
void GUIComponent::InputField::Draw ( 
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Draw the component.

This function draws the component.

#### Parameters

<i>base</i>	The base position of the component.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

### 7.33.4.4 DrawField()

```
void GUIComponent::IntegerInputField::DrawField ( 
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Implements [GUIComponent::InputField](#).

#### 7.33.4.5 ExtractValue()

```
std::string GUIComponent::IntegerField::ExtractValue ( ) [virtual]
```

Implements [GUIComponent::InputField](#).

#### 7.33.4.6 GetEditMode()

```
bool GUIComponent::InputField::GetEditMode ( ) const [protected], [virtual], [inherited]
```

#### 7.33.4.7 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.33.4.8 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

#### 7.33.4.9 GetLabel()

```
std::string GUIComponent::InputField::GetLabel ( ) const [virtual], [inherited]
```

#### 7.33.4.10GetPosition()

```
Vector2 GUI::Component::GetPosition ( ) [inherited]
```

Get the position of the component.

This function gets the position of the component.

##### Returns

The position of the component.

#### 7.33.4.11GetSize()

```
Vector2 GUIComponent::InputField::GetSize ( ) [virtual], [inherited]
```

Get the size of the component.

This function gets the size of the component.

##### Returns

The size of the component.

Reimplemented from [GUI::Component](#).

#### 7.33.4.12GetVisible()

```
bool GUI::Component::GetVisible ( ) [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

##### Returns

The visibility of the component.

#### 7.33.4.13 IsClicked()

```
bool GUIComponent::InputField::IsClicked (
    Vector2 base = (Vector2){0, 0} ) const [protected], [virtual], [inherited]
```

#### 7.33.4.14 isSelectable()

```
bool GUIComponent::IntegerField::isSelectable () const [virtual]
```

Check if the component is selectable.

**Deprecated** This function is deprecated.

This function checks if the component is selectable.

##### Returns

True if the component is selectable.

Reimplemented from [GUIComponent::InputField](#).

#### 7.33.4.15 isSelected()

```
bool GUI::Component::isSelected () const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.33.4.16 Randomize()

```
void GUIComponent::IntegerField::Randomize () [virtual]
```

Reimplemented from [GUIComponent::InputField](#).

#### 7.33.4.17 **select()**

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

#### 7.33.4.18 **SetConstraint()**

```
void GUIComponent::IntegerField::SetConstraint (
    int minValue,
    int maxValue )
```

#### 7.33.4.19 **SetEditMode()**

```
void GUIComponent::InputField::SetEditMode (
    bool canEdit ) [protected], [virtual], [inherited]
```

#### 7.33.4.20 **SetInputFieldSize()**

```
void GUIComponent::InputField::SetInputFieldSize (
    Vector2 size ) [virtual], [inherited]
```

#### 7.33.4.21 **SetLabel()**

```
void GUIComponent::InputField::SetLabel (
    std::string labelContent ) [virtual], [inherited]
```

#### 7.33.4.22 **SetLabelSize()**

```
void GUIComponent::InputField::SetLabelSize (
    float fontSize ) [virtual], [inherited]
```

#### 7.33.4.23 **SetPosition() [1/2]**

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

**7.33.4.24 SetPosition() [2/2]**

```
void GUI::Component::SetPosition (
    Vector2 position ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>position</i>	The position of the component.
-----------------	--------------------------------

**7.33.4.25 SetVisible()**

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

**Parameters**

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

**7.33.4.26 ToggleVisible()**

```
void GUI::Component::ToggleVisible ( ) [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.33.4.27 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.33.4.28 UpdateMouseCursorWhenHover() [2/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

### 7.33.5 Member Data Documentation

#### 7.33.5.1 editMode

```
bool GUIComponent::InputField::editMode [protected], [inherited]
```

#### 7.33.5.2 extractedValue

```
std::string GUIComponent::InputField::extractedValue [protected], [inherited]
```

### 7.33.5.3 fields

```
std::vector< bool > GUIComponent::InputField::fields [static], [inherited]
```

### 7.33.5.4 fonts

```
FontHolder* GUIComponent::InputField::fonts [protected], [inherited]
```

### 7.33.5.5 input

```
int GUIComponent::IntegerField::input [private]
```

### 7.33.5.6 inputFieldSize

```
Vector2 GUIComponent::InputField::inputFieldSize [protected], [inherited]
```

### 7.33.5.7 label

```
std::string GUIComponent::InputField::label [protected], [inherited]
```

### 7.33.5.8 labelFontSize

```
float GUIComponent::InputField::labelFontSize [protected], [inherited]
```

### 7.33.5.9 mFieldIndex

```
std::size_t GUIComponent::InputField::mFieldIndex [protected], [inherited]
```

### 7.33.5.10 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

### 7.33.5.11 m.MaxValue

```
int GUIComponent::IntegerInputField::m.MaxValue [private]
```

### 7.33.5.12 m.MinValue

```
int GUIComponent::IntegerInputField::m.MinValue [private]
```

### 7.33.5.13 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

### 7.33.5.14 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

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

- src/Components/Common/[IntegerInputField.hpp](#)
- src/Components/Common/[IntegerInputField.cpp](#)

## 7.34 Core::List< T >::iterator Class Reference

The list iterator class.

```
#include <List.hpp>
```

## Public Types

- using `iterator_category` = std::bidirectional\_iterator\_tag
- using `value_type` = T
- using `difference_type` = std::ptrdiff\_t
- using `pointer` = T \*
- using `reference` = T &

## Public Member Functions

- `iterator ()`  
*Construct a new iterator object.*
- `iterator (Node< value_type > *const &p)`  
*Construct a new iterator object.*
- `reference operator*& () const`
- `pointer operator-> () const`  
*Returns a pointer to the object pointed to by the iterator.*
- `iterator & operator++ ()`  
*Prefix increment.*
- `iterator operator++ (int)`  
*Postfix increment.*
- `iterator & operator-- ()`  
*Prefix decrement.*
- `iterator operator-- (int)`  
*Postfix decrement.*
- `iterator & operator= (Node< value_type > *const &p)`  
*Assignment operator.*
- `iterator operator+ (const int &step)`  
*Returns an iterator after increased step steps.*
- `iterator operator- (const int &step)`  
*Returns an iterator after decreased step steps.*
- `bool operator== (const iterator &it) const`  
*Checks if two iterators are equal.*
- `bool operator!= (const iterator &it) const`  
*Checks if two iterators are not equal.*
- `void swap (iterator &other)`  
*Swaps the contents of the iterator with those of other.*

## Private Attributes

- `Node< T > * ptr`

## Friends

- class `List`

### 7.34.1 Detailed Description

```
template<typename T>
class Core::List< T >::iterator
```

The list iterator class.

**Template Parameters**

<i>T</i>	The type of the elements
----------	--------------------------

**7.34.2 Member Typedef Documentation****7.34.2.1 difference\_type**

```
template<typename T >
using Core::List< T >::iterator::difference_type = std::ptrdiff_t
```

**7.34.2.2 iterator\_category**

```
template<typename T >
using Core::List< T >::iterator::iterator_category = std::bidirectional_iterator_tag
```

**7.34.2.3 pointer**

```
template<typename T >
using Core::List< T >::iterator::pointer = T*
```

**7.34.2.4 reference**

```
template<typename T >
using Core::List< T >::iterator::reference = T&
```

**7.34.2.5 value\_type**

```
template<typename T >
using Core::List< T >::iterator::value_type = T
```

**7.34.3 Constructor & Destructor Documentation**

### 7.34.3.1 iterator() [1/2]

```
template<typename T >
Core::List< T >::iterator::iterator ( ) [inline]
```

Construct a new iterator object.

### 7.34.3.2 iterator() [2/2]

```
template<typename T >
Core::List< T >::iterator::iterator (
    Node< value_type > *const & p ) [inline]
```

Construct a new iterator object.

#### Parameters

<i>p</i>	The pointer to the node
----------	-------------------------

## 7.34.4 Member Function Documentation

### 7.34.4.1 operator"!=()

```
template<typename T >
bool Core::List< T >::iterator::operator!= (
    const iterator & it ) const [inline]
```

Checks if two iterators are not equal.

#### Parameters

<i>it</i>	The iterator to compare with
-----------	------------------------------

#### Returns

true if the iterators are not equal

### 7.34.4.2 operator\*()

```
template<typename T >
reference Core::List< T >::iterator::operator* ( ) const [inline]
```

#### 7.34.4.3 operator+()

```
template<typename T >
iterator Core::List< T >::iterator::operator+ (
    const int & step ) [inline]
```

Returns an iterator after increased step steps.

##### Parameters

step	The number of steps to increase
------	---------------------------------

##### Returns

the resulting iterator

#### 7.34.4.4 operator++() [1/2]

```
template<typename T >
iterator & Core::List< T >::iterator::operator++ ( ) [inline]
```

Prefix increment.

##### Returns

reference to the incremented iterator

#### 7.34.4.5 operator++() [2/2]

```
template<typename T >
iterator Core::List< T >::iterator::operator++ (
    int ) [inline]
```

Postfix increment.

##### Returns

copy of the iterator before increment

#### 7.34.4.6 operator-()

```
template<typename T >
iterator Core::List< T >::iterator::operator- (
    const int & step ) [inline]
```

Returns an iterator after decreased step steps.

**Parameters**

<code>step</code>	The number of steps to decrease
-------------------	---------------------------------

**Returns**

the resulting iterator

**7.34.4.7 operator--() [1/2]**

```
template<typename T >
iterator & Core::List< T >::iterator::operator-- ( ) [inline]
```

Prefix decrement.

**Returns**

reference to the decremented iterator

**7.34.4.8 operator--() [2/2]**

```
template<typename T >
iterator Core::List< T >::iterator::operator-- (
    int ) [inline]
```

Postfix decrement.

**Returns**

copy of the iterator before decrement

**7.34.4.9 operator->()**

```
template<typename T >
pointer Core::List< T >::iterator::operator-> ( ) const [inline]
```

Returns a pointer to the object pointed to by the iterator.

**Returns**

pointer to the object pointed to by the iterator

**7.34.4.10 operator=( )**

```
template<typename T >
iterator & Core::List< T >::iterator::operator= (
    Node< value_type > *const & p ) [inline]
```

Assignment operator.

**Parameters**

<code>p</code>	The pointer to the node
----------------	-------------------------

**Returns**

reference to the assigned iterator

**7.34.4.11 operator==( )**

```
template<typename T >
bool Core::List< T >::iterator::operator== (
    const iterator & it ) const [inline]
```

Checks if two iterators are equal.

**Parameters**

<code>it</code>	The iterator to compare with
-----------------	------------------------------

**Returns**

true if the iterators are equal

**7.34.4.12 swap()**

```
template<typename T >
void Core::List< T >::iterator::swap (
    iterator & other ) [inline]
```

Swaps the contents of the iterator with those of other.

**Parameters**

<code>other</code>	iterator to exchange the contents with
--------------------	--

**7.34.5 Friends And Related Function Documentation**

### 7.34.5.1 List

```
template<typename T >
friend class List [friend]
```

## 7.34.6 Member Data Documentation

### 7.34.6.1 ptr

```
template<typename T >
Node< T >* Core::List< T >::iterator::ptr [private]
```

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

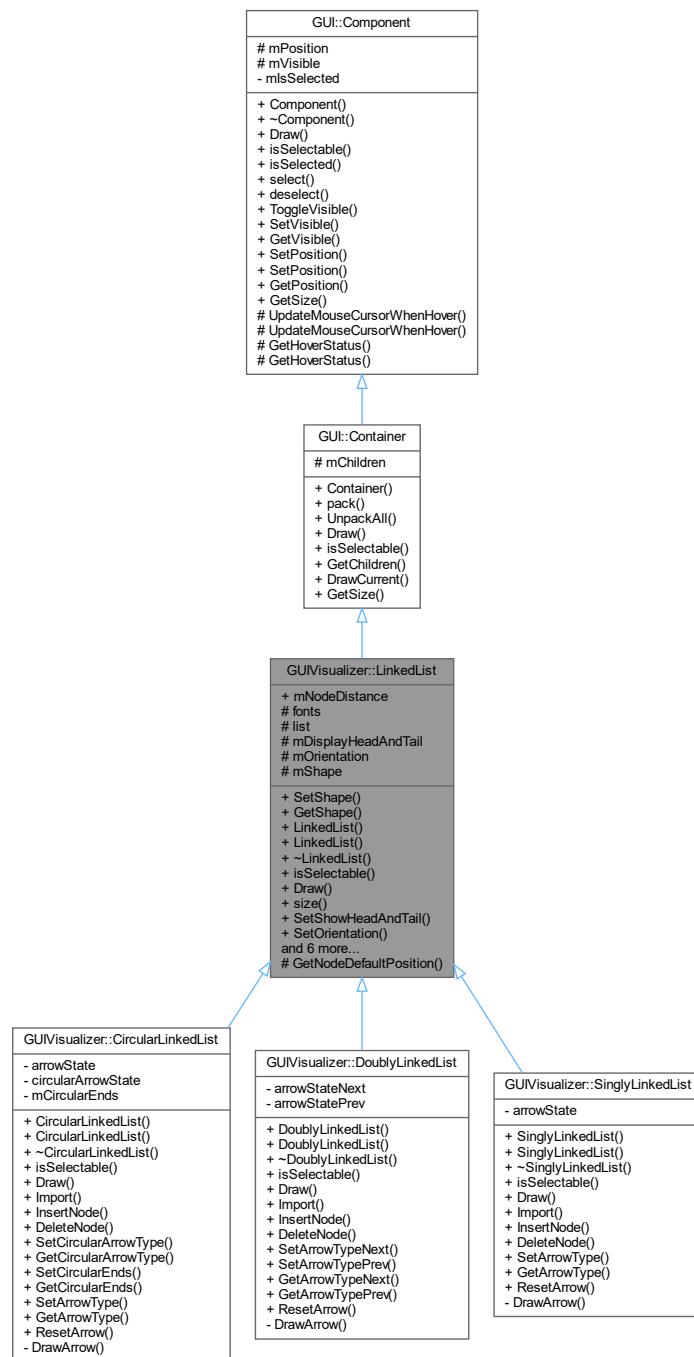
- src/Core/[List.hpp](#)

## 7.35 GUIVisualizer::LinkedList Class Reference

The base class for the linked list visualization. This class provides the basic functionality for the linked list visualization.

```
#include <LinkedList.hpp>
```

Inheritance diagram for GUIVisualizer::LinkedList:



## Public Types

- enum `ArrowType` {
 `Default` , `Hidden` , `Active` , `Skip` ,
 `ArrowTypeCount` }
- The type of the arrow.*
- enum `Orientation` { `Horizontal` , `Vertical` , `OrientationCount` }

- The orientation of the linked list.*
- `typedef std::shared_ptr< Container > Ptr`  
*The shared pointer to the container.*
- ## Public Member Functions
- `void SetShape (Node::Shape shape)`  
*Set the shape of the node.*
  - `Node::Shape GetShape () const`  
*Get the shape of the node.*
  - `LinkedList ()`  
*Construct a new linked list visualization.*
  - `LinkedList (FontHolder *fonts)`  
*Construct a new linked list visualization.*
  - `~LinkedList ()`  
*Destroy the linked list visualization.*
  - `bool IsSelectable () const`  
*Check if the container is selectable.*
  - `virtual void Draw (Vector2 base=(Vector2){0, 0}, float t=1.0f, bool init=false)=0`  
*Draw the linked list visualization.*
  - `virtual std::size_t size () const`  
*Get the size of the linked list visualization.*
  - `virtual void SetShowHeadAndTail (bool show)`
  - `virtual void SetOrientation (Orientation orientation)`  
*Set the orientation of the linked list visualization.*
  - `virtual std::vector< Node > & GetList ()`
  - `virtual Node GenerateNode (int value)`
  - `virtual void Import (std::vector< int > nodes)`  
*Initialize the linked list visualization with the given nodes.*
  - `virtual void InsertNode (std::size_t index, Node node, bool rePosition=true)`  
*Insert a node into the linked list visualization.*
  - `virtual void DeleteNode (std::size_t index, bool rePosition=true)`  
*Delete a node from the singly linked list visualization.*
  - `virtual void Relayout ()`  
*Relayout the linked list visualization.*
  - `void pack (Component::Ptr component)`  
*Pack a component into the container.*
  - `void UnpackAll ()`  
*Unpack all components from the container.*
  - `virtual void Draw (Vector2 base=(Vector2){0, 0})`  
*Draw the container.*
  - `Core::Deque< Component::Ptr > GetChildren ()`  
*Get the pack components of the container.*
  - `virtual void DrawCurrent (Vector2 base)`
  - `virtual Vector2 GetSize ()`  
*Get the size of the container.*
  - `bool IsSelected () const`  
*Check if the component is selected.*
  - `virtual void select ()`  
*Select the component.*

- virtual void **deselect** ()
 

*Deselect the component.*
- virtual void **ToggleVisible** ()
 

*Toggle the visibility of the component.*
- virtual void **SetVisible** (bool visible)
 

*Set the visibility of the component.*
- virtual bool **GetVisible** ()
 

*Get the visibility of the component.*
- void **SetPosition** (float x, float y)
 

*Set the position of the component.*
- void **SetPosition** (Vector2 position)
 

*Set the position of the component.*
- Vector2 **GetPosition** ()
 

*Get the position of the component.*

## Static Public Attributes

- static constexpr float **mNodeDistance** = 20
 

*The distance between the **GUI** nodes.*

## Protected Member Functions

- Vector2 **GetNodeDefaultPosition** (std::size\_t index)
- virtual void **UpdateMouseCursorWhenHover** (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)
 

*Update the mouse cursor when hovering.*
- virtual void **UpdateMouseCursorWhenHover** (Rectangle bound, bool hover, bool noHover)
 

*Update the mouse cursor when hovering.*
- virtual bool **GetHoverStatus** (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)
 

*Get the hover status of the component.*
- virtual bool **GetHoverStatus** (Rectangle bound, bool hover, bool noHover)
 

*Get the hover status of the component.*

## Protected Attributes

- **FontHolder** \* fonts
- std::vector< **Node** > list
- bool **mDisplayHeadAndTail**
- Orientation **mOrientation** = Orientation::Horizontal
 

*The orientation of the linked list, by default it is horizontal.*
- **Node::Shape** **mShape** = **Node::Shape::Circle**

*The shape of the node, by default it is a circle.*
- **Core::Deque**< Component::Ptr > **mChildren**
- Vector2 **mPosition**

*The position of the component.*
- bool **mVisible**

*The visibility of the component.*

## Private Attributes

- bool `mIsSelected`

*The selected status of the component.*

### 7.35.1 Detailed Description

The base class for the linked list visualization. This class provides the basic functionality for the linked list visualization.

### 7.35.2 Member Typedef Documentation

#### 7.35.2.1 Ptr

```
typedef std::shared_ptr< Container > GUI::Container::Ptr [inherited]
```

The shared pointer to the container.

The `GUI` container is commonly used by multiple components, so a shared pointer is used.

See also

```
std::shared_ptr
```

### 7.35.3 Member Enumeration Documentation

#### 7.35.3.1 ArrowType

```
enum GUIVisualizer::LinkedList::ArrowType
```

The type of the arrow.

The type of the arrow is used to determine the color of the arrow.

Enumerator

Default	
Hidden	
Active	
Skip	
ArrowTypeCount	

### 7.35.3.2 Orientation

```
enum GUIVisualizer::LinkedList::Orientation
```

The orientation of the linked list.

The orientation of the linked list is used to determine the orientation of the linked list.

Enumerator

Horizontal	
Vertical	
OrientationCount	

## 7.35.4 Constructor & Destructor Documentation

### 7.35.4.1 LinkedList() [1/2]

```
GUIVisualizer::LinkedList::LinkedList ( )
```

Construct a new linked list visualization.

### 7.35.4.2 LinkedList() [2/2]

```
GUIVisualizer::LinkedList::LinkedList (   
    FontHolder * fonts )
```

Construct a new linked list visualization.

Parameters

<i>fonts</i>	The fonts to be used in the linked list visualization.
--------------	--

### 7.35.4.3 ~LinkedList()

```
GUIVisualizer::LinkedList::~LinkedList ( )
```

Destroy the linked list visualization.

### 7.35.5 Member Function Documentation

#### 7.35.5.1 DeleteNode()

```
void GUIVisualizer::LinkedList::DeleteNode (
    std::size_t index,
    bool rePosition = true ) [virtual]
```

Delete a node from the singly linked list visualization.

##### Parameters

<i>index</i>	The index to delete the node.
<i>rePosition</i>	Whether to reposition the whole singly linked list after deletion.

##### See also

[Relayout](#)

Reimplemented in [GUIVisualizer::CircularLinkedList](#), [GUIVisualizer::DoublyLinkedList](#), and [GUIVisualizer::SinglyLinkedList](#).

#### 7.35.5.2 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

#### 7.35.5.3 Draw() [1/2]

```
void GUI::Container::Draw (
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Draw the container.

This function draws the container.

##### Parameters

<i>base</i>	The base position of the container.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#).

#### 7.35.5.4 Draw() [2/2]

```
virtual void GUIVisualizer::LinkedList::Draw (
    Vector2 base = (Vector2){0, 0},
    float t = 1.0f,
    bool init = false ) [pure virtual]
```

Draw the linked list visualization.

##### Parameters

<i>base</i>	The base position of the linked list visualization.
<i>t</i>	The interpolation value.
<i>init</i>	Whether the linked list visualization is initialized.

This function draws the linked list visualization.

Implemented in [GUIVisualizer::CircularLinkedList](#), [GUIVisualizer::DoublyLinkedList](#), and [GUIVisualizer::SinglyLinkedList](#).

#### 7.35.5.5 DrawCurrent()

```
void GUI::Container::DrawCurrent (
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

#### 7.35.5.6 GenerateNode()

```
GUIVisualizer::Node GUIVisualizer::LinkedList::GenerateNode (
    int value ) [virtual]
```

#### 7.35.5.7 GetChildren()

```
Core::Deque< GUI::Component::Ptr > GUI::Container::GetChildren ( ) [inherited]
```

Get the pack components of the container.

This function returns the all of the pack components of the container.

##### Returns

The children of the container.

### 7.35.5.8 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

#### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

### 7.35.5.9 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

#### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

### 7.35.5.10 GetList()

```
std::vector< GUIVisualizer::Node > & GUIVisualizer::LinkedList::GetList () [virtual]
```

### 7.35.5.11 GetNodeDefaultPosition()

```
Vector2 GUIVisualizer::LinkedList::GetNodeDefaultPosition (
    std::size_t index) [protected]
```

### 7.35.5.12 GetPosition()

```
Vector2 GUI::Component::GetPosition ( ) [inherited]
```

Get the position of the component.

This function gets the position of the component.

#### Returns

The position of the component.

### 7.35.5.13 GetShape()

```
GUIVisualizer::Node::Shape GUIVisualizer::LinkedList::GetShape ( ) const
```

Get the shape of the node.

### 7.35.5.14 GetSize()

```
Vector2 GUI::Container::GetSize ( ) [virtual], [inherited]
```

Get the size of the container.

This function returns the size of the container.

#### Returns

The size of the container.

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#), and [GUIComponent::OptionInputField](#).

### 7.35.5.15 GetVisible()

```
bool GUI::Component::GetVisible ( ) [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

#### Returns

The visibility of the component.

### 7.35.5.16 Import()

```
void GUIVisualizer::LinkedList::Import (
    std::vector< int > nodes ) [virtual]
```

Initialize the linked list visualization with the given nodes.

**Parameters**

<i>nodes</i>	The nodes to initialize the linked list visualization.
--------------	--

Reimplemented in [GUIVisualizer::CircularLinkedList](#), [GUIVisualizer::DoublyLinkedList](#), and [GUIVisualizer::SinglyLinkedList](#).

### 7.35.5.17 InsertNode()

```
void GUIVisualizer::LinkedList::InsertNode (
    std::size_t index,
    GUIVisualizer::Node node,
    bool rePosition = true ) [virtual]
```

Insert a node into the linked list visualization.

**Parameters**

<i>index</i>	The index to insert the node.
<i>node</i>	The node to insert.
<i>rePosition</i>	Whether to reposition the whole linked list after insertion.

**See also**

[Relayout](#)

Reimplemented in [GUIVisualizer::CircularLinkedList](#), [GUIVisualizer::DoublyLinkedList](#), and [GUIVisualizer::SinglyLinkedList](#).

### 7.35.5.18 isSelectable()

```
bool GUIVisualizer::LinkedList::isSelectable () const [virtual]
```

Check if the container is selectable.

**Deprecated** This function is deprecated.

This function checks if the container is selectable.

**Returns**

True if the container is selectable.

Reimplemented from [GUI::Container](#).

Reimplemented in [GUIVisualizer::SinglyLinkedList](#).

### 7.35.5.19 isSelected()

```
bool GUI::Component::isSelected ( ) const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

#### Returns

True if the component is selected.

### 7.35.5.20 pack()

```
void GUI::Container::pack (
    Component::Ptr component ) [inherited]
```

Pack a component into the container.

This function packs a component into the container.

#### Parameters

<i>component</i>	The component to pack.
------------------	------------------------

### 7.35.5.21 Relayout()

```
void GUIVisualizer::LinkedList::Relayout ( ) [virtual]
```

relayout the linked list visualization.

### 7.35.5.22 select()

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

### 7.35.5.23 SetOrientation()

```
void GUIVisualizer::LinkedList::SetOrientation (
    Orientation orientation ) [virtual]
```

Set the orientation of the linked list visualization.

#### Parameters

<i>orientation</i>	The orientation of the linked list visualization.
--------------------	---

### 7.35.5.24 SetPosition() [1/2]

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

#### Parameters

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

### 7.35.5.25 SetPosition() [2/2]

```
void GUI::Component::SetPosition (
    Vector2 position ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

#### Parameters

<i>position</i>	The position of the component.
-----------------	--------------------------------

### 7.35.5.26 SetShape()

```
void GUIVisualizer::LinkedList::SetShape (
    Node::Shape shape )
```

Set the shape of the node.

#### 7.35.5.27 SetShowHeadAndTail()

```
void GUIVisualizer::LinkedList::SetShowHeadAndTail (
    bool show ) [virtual]
```

#### 7.35.5.28 SetVisible()

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

##### Parameters

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

#### 7.35.5.29 size()

```
std::size_t GUIVisualizer::LinkedList::size ( ) const [virtual]
```

Get the size of the linked list visualization.

##### Returns

The size of the linked list visualization.

#### 7.35.5.30 ToggleVisible()

```
void GUI::Component::ToggleVisible ( ) [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

### 7.35.5.31 UnpackAll()

```
void GUI::Container::UnpackAll ( ) [inherited]
```

Unpack all components from the container.

This function unpacks all components from the container.

#### See also

[pack](#)

### 7.35.5.32 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

#### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

### 7.35.5.33 UpdateMouseCursorWhenHover() [2/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

#### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

## 7.35.6 Member Data Documentation

### 7.35.6.1 fonts

```
FontHolder* GUIVisualizer::LinkedList::fonts [protected]
```

### 7.35.6.2 list

```
std::vector< Node > GUIVisualizer::LinkedList::list [protected]
```

### 7.35.6.3 mChildren

```
Core::Deque< Component::Ptr > GUI::Container::mChildren [protected], [inherited]
```

### 7.35.6.4 mDisplayHeadAndTail

```
bool GUIVisualizer::LinkedList::mDisplayHeadAndTail [protected]
```

### 7.35.6.5 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

### 7.35.6.6 mNodeDistance

```
constexpr float GUIVisualizer::LinkedList::mNodeDistance = 20 [static], [constexpr]
```

The distance between the **GUI** nodes.

### 7.35.6.7 mOrientation

```
Orientation GUIVisualizer::LinkedList::mOrientation = Orientation::Horizontal [protected]
```

The orientation of the linked list, by default it is horizontal.

### 7.35.6.8 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

### 7.35.6.9 mShape

```
Node::Shape GUIVisualizer::LinkedList::mShape = Node::Shape::Circle [protected]
```

The shape of the node, by default it is a circle.

### 7.35.6.10 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

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

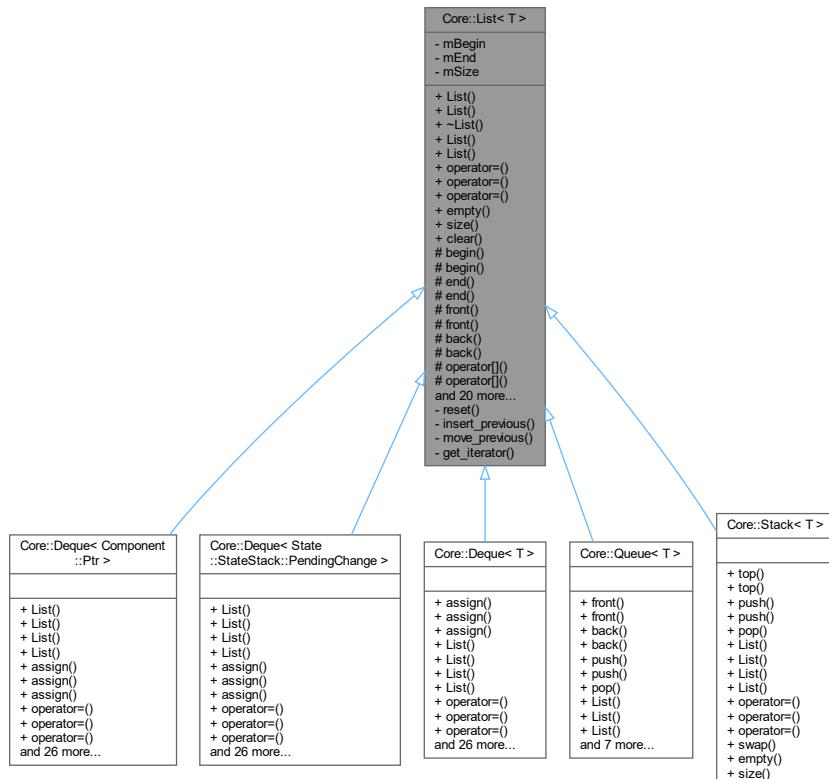
- src/Components/Visualization/[LinkedList.hpp](#)
- src/Components/Visualization/[LinkedList.cpp](#)

## 7.36 Core::List< T > Class Template Reference

The base container for implementing other data structures.

```
#include <List.hpp>
```

Inheritance diagram for Core::List< T >:



## Classes

- class [const\\_iterator](#)  
*The list `const_iterator` class.*
- class [iterator](#)  
*The list `iterator` class.*

## Public Member Functions

- [List \(\)](#)  
*Default constructor.*
- [List \(std::initializer\\_list< T > list\)](#)  
*Constructs the container with the contents of the initializer list.*
- [~List \(\)](#)  
*Destructor.*
- [List \(const List< T > &list\)](#)

- *Copy constructor.*
- `List (List< T > &&list)`
  - *Move constructor.*
- `List< T > & operator= (std::initializer_list< T > list)`
  - *Copy assign the container with the contents of the initializer list.*
- `List< T > & operator= (const List< T > &list)`
  - *Copy assignment operator.*
- `List< T > & operator= (List< T > &&list)`
  - *Move assignment operator.*
- `bool empty () const`
  - *Check whether the container is empty.*
- `std::size_t size () const`
  - *Returns the size of the container.*
- `void clear ()`
  - *Frees all elements in the container.*

## Protected Member Functions

- `iterator begin ()`
  - *Returns the reference to the element at the front of the container.*
- `const_iterator begin () const`
  - *Returns the reference to the element at the front of the container.*
- `iterator end ()`
  - *Returns the reference to the element at the back of the container.*
- `const_iterator end () const`
  - *Returns the reference to the element at the back of the container.*
- `T & front ()`
  - *Returns the reference to the element at the front of the container.*
- `const T & front () const`
  - *Returns the reference to the element at the front of the container.*
- `T & back ()`
  - *Returns the reference to the element at the back of the container.*
- `const T & back () const`
  - *Returns the reference to the element at the back of the container.*
- `T & operator[] (std::size_t index)`
  - *Returns the reference to the element at the given index.*
- `const T & operator[] (std::size_t index) const`
  - *Returns the reference to the element at the given index.*
- `T & at (std::size_t index)`
  - *Returns the reference to the element at the given index.*
- `const T & at (std::size_t index) const`
  - *Returns the reference to the element at the given index.*
- `void push_front (const T &value)`
  - *Pushes the element to the front of the container.*
- `void push_back (const T &value)`
  - *Pushes the element to the back of the container.*
- `void pop_front ()`
  - *Removes the element at the front of the container.*
- `void pop_back ()`
  - *Removes the element at the back of the container.*
- `iterator remove (const iterator &it)`
  - *Removes the element iterator in the container.*
- `int remove (const T &value, const iterator &begin, const iterator &end)`
  - *Removes the elements in the range [begin, end]*

- int `remove` (const T &value)
 

*Removes the elements that has the same value as the given one.*
- int `remove_if` (std::function< bool(const T &) > predicate, const iterator &begin, const iterator &end)
 

*Removes the elements that satisfy the given predicate.*
- int `remove_if` (std::function< bool(const T &) > predicate)
 

*Removes the elements that satisfy the given predicate.*
- void `resize` (std::size\_t count)
 

*Resizes the container to contain the given number of elements.*
- void `resize` (std::size\_t count, const T &value)
 

*Resizes the container to contain the given number of elements.*
- void `assign` (std::size\_t count, const T &value)
 

*Assigns new contents to the list, replacing its current content with count copies of value*
- void `assign` (const const\_iterator &begin, const const\_iterator &end)
 

*Assigns new contents to the list, replacing its current content.*
- void `assign` (std::initializer\_list< T > list)
 

*Assigns new contents to the list, replacing its current content.*
- std::size\_t `unique` ()
 

*Eliminates all except the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.*
- std::size\_t `unique` (std::function< bool(const T &, const T &) > predicate)
 

*Eliminates all except the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.*
- void `reverse` ()
 

*Reverses the order of the elements in the list.*
- void `swap` (List< T > &other)
 

*Swaps the contents of the list with those of other.*

## Private Member Functions

- void `reset` ()
 

*Initializes the list to an empty state.*
- void `insert_previous` (const iterator &it, const iterator &it\_prev)
 

*Inserts an iterator before the specified iterator.*
- iterator `move_previous` (const iterator &it, const iterator &first, const iterator &last)
 

*Moves the iterator to the previous position in the list.*
- iterator `get_iterator` (std::size\_t index)
 

*Returns an iterator to the element at index index*

## Private Attributes

- iterator `mBegin`

*The head of the list.*
- iterator `mEnd`

*The end of the list (one past the last element)*
- std::size\_t `mSize`

*The size of the list.*

### 7.36.1 Detailed Description

```
template<typename T>
class Core::List< T >
```

The base container for implementing other data structures.

**Template Parameters**

<i>T</i>	the type of the elements
----------	--------------------------

## 7.36.2 Constructor & Destructor Documentation

### 7.36.2.1 List() [1/4]

```
template<typename T >
Core::List< T >::List ( ) [inline]
```

Default constructor.

### 7.36.2.2 List() [2/4]

```
template<typename T >
Core::List< T >::List (
    std::initializer_list< T > list ) [inline]
```

Constructs the container with the contents of the initializer list.

**Parameters**

<i>init_list</i>	The initializer list
------------------	----------------------

### 7.36.2.3 ~List()

```
template<typename T >
Core::List< T >::~List ( ) [inline]
```

Destructor.

### 7.36.2.4 List() [3/4]

```
template<typename T >
Core::List< T >::List (
    const List< T > & list ) [inline]
```

Copy constructor.

**Parameters**

<i>rhs</i>	The other container
------------	---------------------

**7.36.2.5 List() [4/4]**

```
template<typename T >
Core::List< T >::List (
    List< T > && list ) [inline]
```

Move constructor.

**Parameters**

<i>rhs</i>	The other container
------------	---------------------

**7.36.3 Member Function Documentation****7.36.3.1 assign() [1/3]**

```
template<typename T >
void Core::List< T >::assign (
    const const_iterator & begin,
    const const_iterator & end ) [inline], [protected]
```

Assigns new contents to the list, replacing its current content.

**Parameters**

<i>begin</i>	The iterator to the first element to be assigned
<i>end</i>	The iterator to the last element to be assigned

**7.36.3.2 assign() [2/3]**

```
template<typename T >
void Core::List< T >::assign (
    std::initializer_list< T > list ) [inline], [protected]
```

Assigns new contents to the list, replacing its current content.

**Parameters**

<i>list</i>	The initializer list to be assigned
-------------	-------------------------------------

**7.36.3.3 assign() [3/3]**

```
template<typename T >
void Core::List< T >::assign (
    std::size_t count,
    const T & value ) [inline], [protected]
```

Assigns new contents to the list, replacing its current content with `count` copies of `value`

**Parameters**

<i>count</i>	The new size of the container
<i>value</i>	The value to be used to fill the container

**7.36.3.4 at() [1/2]**

```
template<typename T >
T & Core::List< T >::at (
    std::size_t index ) [inline], [protected]
```

Returns the reference to the element at the given index.

**Parameters**

<i>index</i>	The index of the element
--------------	--------------------------

**Returns**

`T&` The reference to the element at the given index

**Exceptions**

`std::out_of_range`: index out of range

**7.36.3.5 at() [2/2]**

```
template<typename T >
```

```
const T & Core::List< T >::at (
    std::size_t index ) const [inline], [protected]
```

Returns the reference to the element at the given index.

#### Parameters

<i>index</i>	The index of the element
--------------	--------------------------

#### Returns

T& The reference to the element at the given index

#### Exceptions

<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------

std::out\_of\_range: index out of range

### 7.36.3.6 back() [1/2]

```
template<typename T >
T & Core::List< T >::back () [inline], [protected]
```

Returns the reference to the element at the back of the container.

#### Returns

T& The reference to the element at the back of the container

#### Exceptions

<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------

undefined behavior: null pointer dereference

### 7.36.3.7 back() [2/2]

```
template<typename T >
const T & Core::List< T >::back () const [inline], [protected]
```

Returns the reference to the element at the back of the container.

#### Returns

T& The reference to the element at the back of the container

**Exceptions**

undefined behavior: null pointer dereference

**7.36.3.8 begin() [1/2]**

```
template<typename T >
iterator Core::List< T >::begin ( )  [inline], [protected]
```

**7.36.3.9 begin() [2/2]**

```
template<typename T >
const_iterator Core::List< T >::begin ( ) const  [inline], [protected]
```

**7.36.3.10 clear()**

```
template<typename T >
void Core::List< T >::clear ( )  [inline]
```

Frees all elements in the container.

**7.36.3.11 empty()**

```
template<typename T >
bool Core::List< T >::empty ( ) const  [inline]
```

Check whether the container is empty.

**Return values**

<i>true</i>	The container is empty
<i>false</i>	The container is not empty

**7.36.3.12 end() [1/2]**

```
template<typename T >
iterator Core::List< T >::end ( )  [inline], [protected]
```

### 7.36.3.13 `end()` [2/2]

```
template<typename T >
const_iterator Core::List< T >::end ( ) const [inline], [protected]
```

### 7.36.3.14 `front()` [1/2]

```
template<typename T >
T & Core::List< T >::front ( ) [inline], [protected]
```

Returns the reference to the element at the front of the container.

#### Returns

T& The reference to the element at the front of the container

#### Exceptions



undefined behavior: null pointer dereference

### 7.36.3.15 `front()` [2/2]

```
template<typename T >
const T & Core::List< T >::front ( ) const [inline], [protected]
```

Returns the reference to the element at the front of the container.

#### Returns

T& The reference to the element at the front of the container

#### Exceptions



undefined behavior: null pointer dereference

### 7.36.3.16 `get_iterator()`

```
template<typename T >
iterator Core::List< T >::get_iterator (
    std::size_t index ) [inline], [private]
```

Returns an iterator to the element at index `index`

#### Parameters

<code>index</code>	The index of the element
--------------------	--------------------------

#### Returns

An iterator to the element at index `index`

#### Exceptions

<code>std::out_of_range</code>	if `index` is out of range
--------------------------------	----------------------------

### 7.36.3.17 `insert_previous()`

```
template<typename T >
void Core::List< T >::insert_previous (
    const iterator & it,
    const iterator & it_prev ) [inline], [private]
```

Insert an iterator before the specified iterator.

#### Parameters

<code>it</code>	The iterator to insert the iterator (prev) before
<code>it_prev</code>	The iterator to insert

### 7.36.3.18 `move_previous()`

```
template<typename T >
iterator Core::List< T >::move_previous (
    const iterator & it,
    const iterator & first,
    const iterator & last ) [inline], [private]
```

### 7.36.3.19 `operator=()` [1/3]

```
template<typename T >
List< T > & Core::List< T >::operator= (
    const List< T > & list ) [inline]
```

Copy assignment operator.

**Parameters**

<i>rhs</i>	The other container
------------	---------------------

**7.36.3.20 operator=() [2/3]**

```
template<typename T >
List< T > & Core::List< T >::operator= (
    List< T > && list ) [inline]
```

Move assignment operator.

**Parameters**

<i>rhs</i>	The other container
------------	---------------------

**7.36.3.21 operator=() [3/3]**

```
template<typename T >
List< T > & Core::List< T >::operator= (
    std::initializer_list< T > list ) [inline]
```

Copy assign the container with the contents of the initializer list.

**Parameters**

<i>init_list</i>	The initializer list
------------------	----------------------

**7.36.3.22 operator[]() [1/2]**

```
template<typename T >
T & Core::List< T >::operator[] (
    std::size_t index ) [inline], [protected]
```

Returns the reference to the element at the given index.

**Parameters**

<i>index</i>	The index of the element
--------------	--------------------------

**Returns**

T& The reference to the element at the given index

**Exceptions**

std::out\_of\_range: index out of range

**7.36.3.23 operator[]( ) [2/2]**

```
template<typename T >
const T & Core::List< T >::operator[] (
    std::size_t index) const [inline], [protected]
```

Returns the reference to the element at the given index.

**Parameters**

<i>index</i>	The index of the element
--------------	--------------------------

**Returns**

T& The reference to the element at the given index

**Exceptions**

std::out\_of\_range: index out of range

**7.36.3.24 pop\_back()**

```
template<typename T >
void Core::List< T >::pop_back () [inline], [protected]
```

Removes the element at the back of the container.

**7.36.3.25 pop\_front()**

```
template<typename T >
void Core::List< T >::pop_front () [inline], [protected]
```

Removes the element at the front of the container.

### 7.36.3.26 push\_back()

```
template<typename T >
void Core::List< T >::push_back (
    const T & value ) [inline], [protected]
```

Pushes the element to the back of the container.

#### Parameters

<i>elem</i>	The element to be pushed into the back
-------------	--

### 7.36.3.27 push\_front()

```
template<typename T >
void Core::List< T >::push_front (
    const T & value ) [inline], [protected]
```

Pushes the element to the front of the container.

#### Parameters

<i>elem</i>	The element to be pushed into the front
-------------	---

### 7.36.3.28 remove() [1/3]

```
template<typename T >
iterator Core::List< T >::remove (
    const iterator & it ) [inline], [protected]
```

Removes the element iterator in the container.

#### Parameters

<i>it</i>	The iterator to the element to be removed
-----------	---

#### Returns

iterator The iterator to the next element

#### Exceptions



std::out\_of\_range: iterator out of range

### 7.36.3.29 remove() [2/3]

```
template<typename T >
int Core::List< T >::remove (
    const T & value ) [inline], [protected]
```

Removes the elements that has the same value as the given one.

#### Parameters

<i>value</i>	The value of the elements to be removed
--------------	---

#### Returns

iterator The iterator to the next element

### 7.36.3.30 remove() [3/3]

```
template<typename T >
int Core::List< T >::remove (
    const T & value,
    const iterator & begin,
    const iterator & end ) [inline], [protected]
```

Removes the elements in the range [begin, end)

#### Parameters

<i>begin</i>	The iterator to the first element to be removed
<i>end</i>	The iterator to the last element to be removed

#### Returns

iterator The iterator to the next element

#### Exceptions



std::out\_of\_range: iterator out of range

### 7.36.3.31 remove\_if() [1/2]

```
template<typename T >
int Core::List< T >::remove_if (
    std::function< bool(const T &) > predicate ) [inline], [protected]
```

### 7.36.3.32 `remove_if()` [2/2]

```
template<typename T >
int Core::List< T >::remove_if (
    std::function< bool(const T &) > predicate,
    const iterator & begin,
    const iterator & end ) [inline], [protected]
```

### 7.36.3.33 `reset()`

```
template<typename T >
void Core::List< T >::reset () [inline], [private]
```

### 7.36.3.34 `resize()` [1/2]

```
template<typename T >
void Core::List< T >::resize (
    std::size_t count ) [inline], [protected]
```

Resizes the container to contain the given number of elements.

#### Parameters

<i>count</i>	The new size of the container
--------------	-------------------------------

### 7.36.3.35 `resize()` [2/2]

```
template<typename T >
void Core::List< T >::resize (
    std::size_t count,
    const T & value ) [inline], [protected]
```

Resizes the container to contain the given number of elements.

#### Parameters

<i>count</i>	The new size of the container
<i>value</i>	The value to be used to fill the container

### 7.36.3.36 reverse()

```
template<typename T >
void Core::List< T >::reverse ( ) [inline], [protected]
```

Reverses the order of the elements in the list.

### 7.36.3.37 size()

```
template<typename T >
std::size_t Core::List< T >::size ( ) const [inline]
```

Returns the size of the container.

#### Returns

The size of the container

### 7.36.3.38 swap()

```
template<typename T >
void Core::List< T >::swap (
    List< T > & other ) [inline], [protected]
```

Swaps the contents of the list with those of other.

#### Parameters

<i>other</i>	list to exchange the contents with
--------------	------------------------------------

### 7.36.3.39 unique() [1/2]

```
template<typename T >
std::size_t Core::List< T >::unique ( ) [inline], [protected]
```

Eliminates all except the first element from every consecutive group of equivalent elements from the range [*first*, *last*) and returns a past-the-end iterator for the new logical end of the range.

#### Returns

the resulting size

### 7.36.3.40 unique() [2/2]

```
template<typename T >
std::size_t Core::List< T >::unique (
    std::function< bool(const T &, const T &) > predicate ) [inline], [protected]
```

Eliminates all except the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.

#### Parameters

<i>predicate</i>	Binary predicate that, taking two values of the same type than those contained in the list, returns true to remove the element passed as first argument from the container, and false otherwise.
------------------	--

#### Returns

the resulting size

## 7.36.4 Member Data Documentation

### 7.36.4.1 mBegin

```
template<typename T >
iterator Core::List< T >::mBegin [private]
```

The head of the list.

### 7.36.4.2 mEnd

```
template<typename T >
iterator Core::List< T >::mEnd [private]
```

The end of the list (one past the last element)

### 7.36.4.3 mSize

```
template<typename T >
std::size_t Core::List< T >::mSize [private]
```

The size of the list.

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

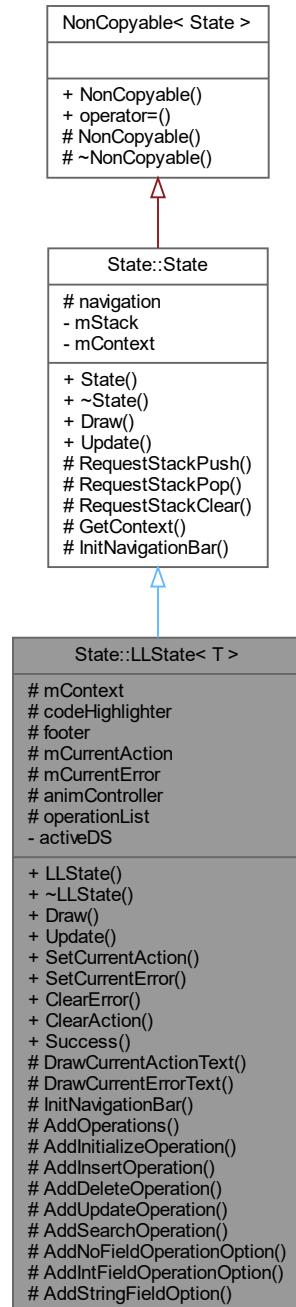
- src/Core/[List.hpp](#)

## 7.37 State::LLState< T > Class Template Reference

The state class that is used as a base linked list state/scene of the application.

```
#include <LLState.hpp>
```

Inheritance diagram for State::LLState< T >:



### Classes

- struct [IntegerInput](#)

## Public Types

- `typedef std::unique_ptr< State > Ptr`  
*A unique pointer to the state object.*

## Public Member Functions

- `LLState (StateStack &stack, Context context, DataStructures::ID activeDS)`  
*Construct a new `LLState` object.*
- `~LLState ()`  
*Destroy the `LLState` object.*
- `virtual void Draw ()`  
*Draw the linked list state to the screen.*
- `virtual bool Update (float dt)`  
*Update the linked list state.*
- `virtual void SetCurrentAction (std::string action)`  
*Set the current action text.*
- `virtual void SetCurrentError (std::string error)`  
*Set the current error text.*
- `virtual void ClearError ()`  
*Clear the current error text.*
- `virtual void ClearAction ()`  
*Clear the current action text.*
- `virtual void Success ()`  
*Clear the current error and toggle the action list.*

## Protected Member Functions

- `virtual void DrawCurrentActionText ()`
- `virtual void DrawCurrentErrorText ()`
- `void InitNavigationBar ()`
- `virtual void AddOperations ()`  
*Add the operations to the operation list.*
- `virtual void AddInitializeOperation ()`  
*Add the initialize operation to the operation list.*
- `virtual void AddInsertOperation ()`  
*Add the insert operation to the operation list.*
- `virtual void AddDeleteOperation ()`  
*Add the delete operation to the operation list.*
- `virtual void AddUpdateOperation ()`  
*Add the update operation to the operation list.*
- `virtual void AddSearchOperation ()`  
*Add the search operation to the operation list.*
- `virtual void AddNoFieldOperationOption (GUIComponent::OperationContainer::Ptr container, std::string title, std::function< void() > action)`
- `virtual void AddIntFieldOperationOption (GUIComponent::OperationContainer::Ptr container, std::string title, Core::Deque< IntegerInput > fields, std::function< void(std::map< std::string, std::string >) > action)`
- `virtual void AddStringFieldOption (GUIComponent::OperationContainer::Ptr container, std::string title, std::string label, std::function< void(std::map< std::string, std::string >) > action)`
- `void RequestStackPush (States::ID stateID)`

- Request the state stack to push a new state/scene.
  - void [RequestStackPop \(\)](#)  
Request the state stack to pop the current state/scene.
  - void [RequestStackClear \(\)](#)  
Request the state stack to clear all the states/scenes.
- [Context GetContext \(\) const](#)  
Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

## Protected Attributes

- [Context mContext](#)
- [GUIComponent::CodeHighlighter::Ptr codeHighlighter](#)
- [GUIComponent::Footer< T > footer](#)
- [std::string mCurrentAction](#)
- [std::string mCurrentError](#)
- [T::Ptr animController](#)
- [GUIComponent::OperationList operationList](#)
- [GUIComponent::NavigationBar navigation](#)

## Private Attributes

- [DataStructures::ID activeDS](#)
- [StateStack \\* mStack](#)

*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*

### 7.37.1 Detailed Description

```
template<typename T>
class State::LLState< T >
```

The state class that is used as a base linked list state/scene of the application.

### 7.37.2 Member Typedef Documentation

#### 7.37.2.1 Ptr

```
typedef std::unique_ptr< State > State::State::Ptr [inherited]
```

A unique pointer to the state object.

### 7.37.3 Constructor & Destructor Documentation

### 7.37.3.1 LLState()

```
template<typename T >
State::LLState< T >::LLState (
    StateStack & stack,
    Context context,
    DataStructures::ID activeDS )
```

Construct a new [LLState](#) object.

**Parameters**

<i>stack</i>	The state stack where the linked list state is pushed to.
<i>context</i>	The context of the application.
<i>activeDS</i>	The active data structure.

**7.37.3.2 ~LLState()**

```
template<typename T >
State::LLState< T >::~LLState
```

Destroy the [LLState](#) object.

**7.37.4 Member Function Documentation****7.37.4.1 AddDeleteOperation()**

```
template<typename T >
void State::LLState< T >::AddDeleteOperation [protected], [virtual]
```

Add the delete operation to the operation list.

Reimplemented in [State::CLLState](#), [State::DLLState](#), [State::QueueState](#), [State::SLLState](#), and [State::StackState](#).

**7.37.4.2 AddInitializeOperation()**

```
template<typename T >
void State::LLState< T >::AddInitializeOperation [protected], [virtual]
```

Add the initialize operation to the operation list.

Reimplemented in [State::CLLState](#), [State::DLLState](#), [State::QueueState](#), [State::SLLState](#), and [State::StackState](#).

**7.37.4.3 AddInsertOperation()**

```
template<typename T >
void State::LLState< T >::AddInsertOperation [protected], [virtual]
```

Add the insert operation to the operation list.

Reimplemented in [State::CLLState](#), [State::DLLState](#), [State::QueueState](#), [State::SLLState](#), and [State::StackState](#).

#### 7.37.4.4 AddIntFieldOperationOption()

```
template<typename T >
void State::LLState< T >::AddIntFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    Core::Deque< IntegerInput > fields,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual]
```

#### 7.37.4.5 AddNoFieldOperationOption()

```
template<typename T >
void State::LLState< T >::AddNoFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::function< void() > action ) [protected], [virtual]
```

#### 7.37.4.6 AddOperations()

```
template<typename T >
void State::LLState< T >::AddOperations [protected], [virtual]
```

Add the operations to the operation list.

##### Note

This function should not be overridden as it calls the other Add\*Operation functions.

#### 7.37.4.7 AddSearchOperation()

```
template<typename T >
void State::LLState< T >::AddSearchOperation [protected], [virtual]
```

Add the search operation to the operation list.

Reimplemented in [State::CLLState](#), [State::DLLState](#), [State::QueueState](#), [State::SLLState](#), and [State::StackState](#).

#### 7.37.4.8 AddStringFieldOption()

```
template<typename T >
void State::LLState< T >::AddStringFieldOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::string label,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual]
```

#### 7.37.4.9 AddUpdateOperation()

```
template<typename T >
void State::LLState< T >::AddUpdateOperation [protected], [virtual]
```

Add the update operation to the operation list.

Reimplemented in [State::CLLState](#), [State::DLLState](#), and [State::SLLState](#).

#### 7.37.4.10 ClearAction()

```
template<typename T >
void State::LLState< T >::ClearAction [inline], [virtual]
```

Clear the current action text.

#### 7.37.4.11 ClearError()

```
template<typename T >
void State::LLState< T >::ClearError [inline], [virtual]
```

Clear the current error text.

#### 7.37.4.12 Draw()

```
template<typename T >
void State::LLState< T >::Draw [inline], [virtual]
```

Draw the linked list state to the screen.

Implements [State::State](#).

#### 7.37.4.13 DrawCurrentActionText()

```
template<typename T >
void State::LLState< T >::DrawCurrentActionText [inline], [protected], [virtual]
```

#### 7.37.4.14 DrawCurrentErrorText()

```
template<typename T >
void State::LLState< T >::DrawCurrentErrorText [inline], [protected], [virtual]
```

#### 7.37.4.15 GetContext()

```
State::State::Context State::State::GetContext () const [protected], [inherited]
```

Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### Returns

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### See also

[Context](#)

#### 7.37.4.16 InitNavigationBar()

```
template<typename T >
void State::LLState< T >::InitNavigationBar [protected]
```

#### 7.37.4.17 RequestStackClear()

```
void State::State::RequestStackClear () [protected], [inherited]
```

Request the state stack to clear all the states/scenes.

##### See also

[StateStack::ClearStates](#)

#### 7.37.4.18 RequestStackPop()

```
void State::State::RequestStackPop ( ) [protected], [inherited]
```

Request the state stack to pop the current state/scene.

See also

[StateStack::PopState](#)

#### 7.37.4.19 RequestStackPush()

```
void State::State::RequestStackPush (
    States::ID stateID ) [protected], [inherited]
```

Request the state stack to push a new state/scene.

Parameters

<i>stateID</i>	The ID of the state/scene that will be pushed
----------------	---

See also

[StateStack::PushState](#)

#### 7.37.4.20 SetCurrentAction()

```
template<typename T >
void State::LLState< T >::SetCurrentAction (
    std::string action ) [inline], [virtual]
```

Set the current action text.

Parameters

<i>action</i>	The current action text.
---------------	--------------------------

#### 7.37.4.21 SetCurrentError()

```
template<typename T >
void State::LLState< T >::SetCurrentError (
    std::string error ) [inline], [virtual]
```

Set the current error text.

**Parameters**

<i>error</i>	The current error text.
--------------	-------------------------

**7.37.4.22 Success()**

```
template<typename T >
void State::LLState< T >::Success [inline], [virtual]
```

Clear the current error and toggle the action list.

**7.37.4.23 Update()**

```
template<typename T >
bool State::LLState< T >::Update (
    float dt ) [virtual]
```

Update the linked list state.

**Parameters**

<i>dt</i>	The delta time between two frames.
-----------	------------------------------------

**Returns**

true If the update was successful.  
false If the update was unsuccessful.

Implements [State::State](#).

**7.37.5 Member Data Documentation****7.37.5.1 activeDS**

```
template<typename T >
DataStructures::ID State::LLState< T >::activeDS [private]
```

### 7.37.5.2 animController

```
template<typename T >
T::Ptr State::LLState< T >::animController [protected]
```

### 7.37.5.3 codeHighlighter

```
template<typename T >
GUIComponent::CodeHighlighter::Ptr State::LLState< T >::codeHighlighter [protected]
```

### 7.37.5.4 footer

```
template<typename T >
GUIComponent::Footer< T > State::LLState< T >::footer [protected]
```

### 7.37.5.5 mContext

```
template<typename T >
Context State::LLState< T >::mContext [protected]
```

### 7.37.5.6 mCurrentAction

```
template<typename T >
std::string State::LLState< T >::mCurrentAction [protected]
```

### 7.37.5.7 mCurrentError

```
template<typename T >
std::string State::LLState< T >::mCurrentError [protected]
```

### 7.37.5.8 mStack

```
StateStack* State::State::mStack [private], [inherited]
```

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

### 7.37.5.9 navigation

```
GUIComponent::NavigationBar State::State::navigation [protected], [inherited]
```

### 7.37.5.10 operationList

```
template<typename T >  
GUIComponent::OperationList State::LLState< T >::operationList [protected]
```

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

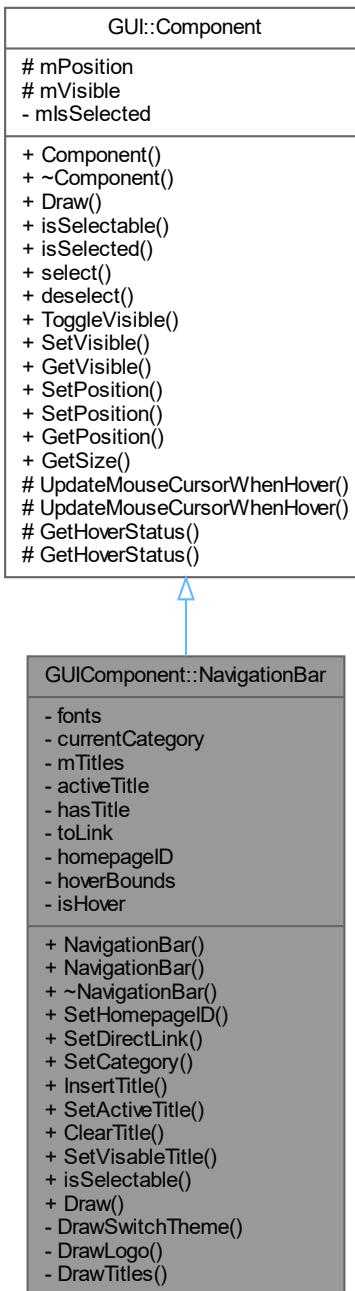
- src/States/LinkedList/[LLState.hpp](#)

## 7.38 GUIComponent::NavigationBar Class Reference

The navigation bar class that is used to represent a navigation bar in the [GUI](#).

```
#include <NavigationBar.hpp>
```

Inheritance diagram for GUIComponent::NavigationBar:



## Classes

- struct [TitleInfo](#)

## Public Types

- `typedef std::shared_ptr<Component> Ptr`  
*The shared pointer to the component.*

## Public Member Functions

- `NavigationBar (FontHolder *fonts)`
- `NavigationBar ()`
- `~NavigationBar ()`
- `void SetHomepageID (States::ID id)`
- `void SetDirectLink (std::function< void(States::ID) > link)`
- `void SetCategory (std::string category)`
- `void InsertTitle (DataStructures::ID titleID, States::ID stateID, std::string abbrTitle, std::string titleName)`
- `void SetActiveTitle (DataStructures::ID title)`
- `void ClearTitle ()`
- `void SetVisibleTitle (bool visible)`
- `bool isSelectable () const`

*Check if the component is selectable.*
- `void Draw (Vector2 base=(Vector2){0, 0})`

*Draw the component.*
- `bool isSelected () const`

*Check if the component is selected.*
- `virtual void select ()`

*Select the component.*
- `virtual void deselect ()`

*Deselect the component.*
- `virtual void ToggleVisible ()`

*Toggle the visibility of the component.*
- `virtual void SetVisible (bool visible)`

*Set the visibility of the component.*
- `virtual bool GetVisible ()`

*Get the visibility of the component.*
- `void SetPosition (float x, float y)`

*Set the position of the component.*
- `void SetPosition (Vector2 position)`

*Set the position of the component.*
- `Vector2 GetPosition ()`

*Get the position of the component.*
- `virtual Vector2 GetSize ()`

*Get the size of the component.*

## Protected Member Functions

- `virtual void UpdateMouseCursorWhenHover (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`

*Update the mouse cursor when hovering.*
- `virtual void UpdateMouseCursorWhenHover (Rectangle bound, bool hover, bool noHover)`

*Update the mouse cursor when hovering.*
- `virtual bool GetHoverStatus (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`

*Get the hover status of the component.*
- `virtual bool GetHoverStatus (Rectangle bound, bool hover, bool noHover)`

*Get the hover status of the component.*

## Protected Attributes

- Vector2 `mPosition`  
*The position of the component.*
- bool `mVisible`  
*The visibility of the component.*

## Private Member Functions

- bool `DrawSwitchTheme ()`
- bool `DrawLogo ()`
- `States::ID DrawTitles ()`

## Private Attributes

- `FontHolder * fonts`
- std::string `currentCategory`
- std::map< `DataStructures::ID`, `TitleInfo` > `mTitles`
- `DataStructures::ID activeTitle`
- bool `hasTitle`
- std::function< void(`States::ID`) > `toLink`
- `States::ID homepageID`
- std::map< std::string, Rectangle > `hoverBounds`
- bool `isHover`
- bool `isSelected`

*The selected status of the component.*

### 7.38.1 Detailed Description

The navigation bar class that is used to represent a navigation bar in the [GUI](#).

The navigation bar is used to navigate between different pages in the application. Also, it have a `switch` theme button that is used to switch between different themes.

### 7.38.2 Member Typedef Documentation

#### 7.38.2.1 Ptr

```
typedef std::shared_ptr< Component > GUI::Component::Ptr [inherited]
```

The shared pointer to the component.

The [GUI](#) component is commonly used by multiple components, so a shared pointer is used.

#### See also

`std::shared_ptr`

### 7.38.3 Constructor & Destructor Documentation

#### 7.38.3.1 NavigationBar() [1/2]

```
GUIComponent::NavigationBar::NavigationBar (
    FontHolder * fonts )
```

#### 7.38.3.2 NavigationBar() [2/2]

```
GUIComponent::NavigationBar::NavigationBar ( )
```

#### 7.38.3.3 ~NavigationBar()

```
GUIComponent::NavigationBar::~NavigationBar ( )
```

### 7.38.4 Member Function Documentation

#### 7.38.4.1 ClearTitle()

```
void GUIComponent::NavigationBar::ClearTitle ( )
```

#### 7.38.4.2 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

#### 7.38.4.3 Draw()

```
void GUIComponent::NavigationBar::Draw (
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Draw the component.

This function draws the component.

**Parameters**

<i>base</i>	The base position of the component.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

#### 7.38.4.4 DrawLogo()

```
bool GUIComponent::NavigationBar::DrawLogo () [private]
```

#### 7.38.4.5 DrawSwitchTheme()

```
bool GUIComponent::NavigationBar::DrawSwitchTheme () [private]
```

#### 7.38.4.6 DrawTitles()

```
States::ID GUIComponent::NavigationBar::DrawTitles () [private]
```

#### 7.38.4.7 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.38.4.8 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
```

```
    bool hover,  
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

#### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

#### 7.38.4.9 GetPosition()

```
Vector2 GUI::Component::GetPosition () [inherited]
```

Get the position of the component.

This function gets the position of the component.

#### Returns

The position of the component.

#### 7.38.4.10 GetSize()

```
Vector2 GUI::Component::GetSize () [virtual], [inherited]
```

Get the size of the component.

This function gets the size of the component.

#### Returns

The size of the component.

Reimplemented in [GUIComponent::Button](#), [GUIComponent::InputField](#), [GUIComponent::OperationList](#), [GUIComponent::OptionInputField](#) and [GUI::Container](#).

#### 7.38.4.11 GetVisible()

```
bool GUI::Component::GetVisible () [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

#### Returns

The visibility of the component.

#### 7.38.4.12 InsertTitle()

```
void GUIComponent::NavigationBar::InsertTitle (
    DataStructures::ID titleID,
    States::ID stateID,
    std::string abbrTitle,
    std::string titleName )
```

#### 7.38.4.13 isSelectable()

```
bool GUIComponent::NavigationBar::isSelectable () const [virtual]
```

Check if the component is selectable.

**Deprecated** This function is deprecated.

This function checks if the component is selectable.

##### Returns

True if the component is selectable.

Implements [GUI::Component](#).

#### 7.38.4.14 isSelected()

```
bool GUI::Component::isSelected () const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.38.4.15 select()

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

#### 7.38.4.16 SetActiveTitle()

```
void GUIComponent::NavigationBar::SetActiveTitle (
    DataStructures::ID title )
```

#### 7.38.4.17 SetCategory()

```
void GUIComponent::NavigationBar::SetCategory (
    std::string category )
```

#### 7.38.4.18 SetDirectLink()

```
void GUIComponent::NavigationBar::SetDirectLink (
    std::function< void(States::ID) > link )
```

#### 7.38.4.19 SetHomepageID()

```
void GUIComponent::NavigationBar::SetHomepageID (
    States::ID id )
```

#### 7.38.4.20 SetPosition() [1/2]

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

**7.38.4.21 SetPosition() [2/2]**

```
void GUI::Component::SetPosition (
    Vector2 position )  [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>position</i>	The position of the component.
-----------------	--------------------------------

**7.38.4.22 SetVisableTitle()**

```
void GUIComponent::NavigationBar::SetVisableTitle (
    bool visible )
```

**7.38.4.23 SetVisible()**

```
void GUI::Component::SetVisible (
    bool visible )  [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

**Parameters**

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

#### 7.38.4.24 ToggleVisible()

```
void GUI::Component::ToggleVisible () [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.38.4.25 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.38.4.26 UpdateMouseCursorWhenHover() [2/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

### 7.38.5 Member Data Documentation

#### 7.38.5.1 activeTitle

```
DataStructures::ID GUIComponent::NavigationBar::activeTitle [private]
```

### 7.38.5.2 currentCategory

```
std::string GUIComponent::NavigationBar::currentCategory [private]
```

### 7.38.5.3 fonts

```
FontHolder* GUIComponent::NavigationBar::fonts [private]
```

### 7.38.5.4 hasTitle

```
bool GUIComponent::NavigationBar::hasTitle [private]
```

### 7.38.5.5 homepageID

```
States::ID GUIComponent::NavigationBar::homepageID [private]
```

### 7.38.5.6 hoverBounds

```
std::map< std::string, Rectangle > GUIComponent::NavigationBar::hoverBounds [private]
```

### 7.38.5.7 isHover

```
bool GUIComponent::NavigationBar::isHover [private]
```

### 7.38.5.8 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

### 7.38.5.9 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

### 7.38.5.10 mTitles

```
std::map< DataStructures::ID, TitleInfo > GUIComponent::NavigationBar::mTitles [private]
```

### 7.38.5.11 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

### 7.38.5.12 toLink

```
std::function< void(States::ID) > GUIComponent::NavigationBar::toLink [private]
```

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

- src/Components/Common/[NavigationBar.hpp](#)
- src/Components/Common/[NavigationBar.cpp](#)

## 7.39 Core::Node< T > Class Template Reference

The node class that is used to store the value of the node, and the pointers to the previous and the next node (similar to Doubly Linked [List](#) node).

```
#include <Node.hpp>
```

### Public Member Functions

- [Node \(\)](#)  
*The default constructor of the node.*
- [Node \(const T &value\)](#)  
*The constructor of the node.*
- [Node \(const Node< T > &node\)](#)  
*The copy constructor of the node.*
- [Node \(const T &value, Node< T > \\*const &prev, Node< T > \\*const &next\)](#)  
*The constructor of the node.*

## Public Attributes

- `T mValue`  
*The value of the node.*
- `Node< T > * mPrev`  
*The pointer to the previous node.*
- `Node< T > * mNext`  
*The pointer to the next node.*

### 7.39.1 Detailed Description

```
template<typename T>
class Core::Node< T >
```

The node class that is used to store the value of the node, and the pointers to the previous and the next node (similar to Doubly Linked [List](#) node).

#### Template Parameters

<code>T</code>	The type of the value of the node.
----------------	------------------------------------

### 7.39.2 Constructor & Destructor Documentation

#### 7.39.2.1 Node() [1/4]

```
template<typename T >
Core::Node< T >::Node ( ) [inline]
```

The default constructor of the node.

#### 7.39.2.2 Node() [2/4]

```
template<typename T >
Core::Node< T >::Node (
    const T & value ) [inline]
```

The constructor of the node.

#### Parameters

<code>value</code>	The value of the node.
--------------------	------------------------

### 7.39.2.3 Node() [3/4]

```
template<typename T >
Core::Node< T >::Node (
    const Node< T > & node ) [inline]
```

The copy constructor of the node.

#### Parameters

<i>node</i>	The node that will be copied.
-------------	-------------------------------

### 7.39.2.4 Node() [4/4]

```
template<typename T >
Core::Node< T >::Node (
    const T & value,
    Node< T > *const & prev,
    Node< T > *const & next ) [inline]
```

The constructor of the node.

#### Parameters

<i>value</i>	The value of the node.
<i>prev</i>	The pointer to the previous node.
<i>next</i>	The pointer to the next node.

## 7.39.3 Member Data Documentation

### 7.39.3.1 mNext

```
template<typename T >
Node< T >* Core::Node< T >::mNext
```

The pointer to the next node.

### 7.39.3.2 mPrev

```
template<typename T >
Node< T *>* Core::Node< T >::mPrev
```

The pointer to the previous node.

### 7.39.3.3 mValue

```
template<typename T >
T Core::Node< T >::mValue
```

The value of the node.

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

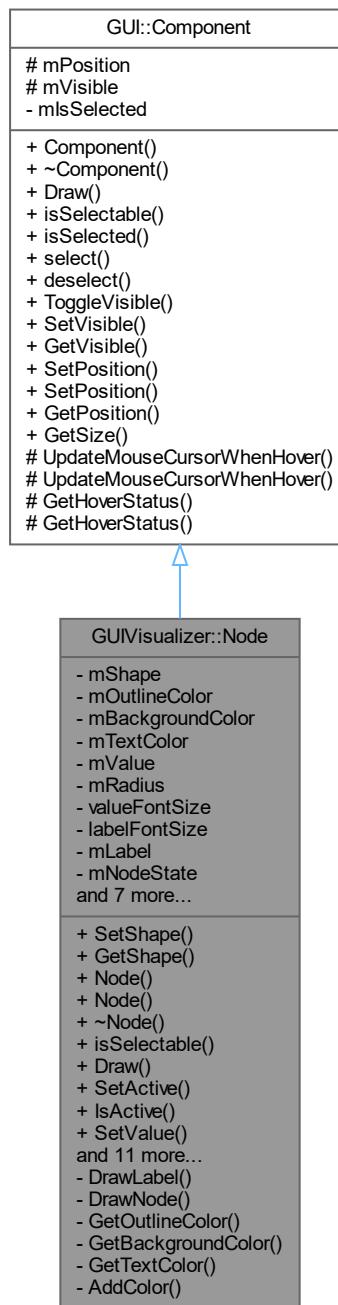
- src/Core/[Node.hpp](#)

## 7.40 GUIVisualizer::Node Class Reference

The node class that is used to represent a node in the visualization.

```
#include <Node.hpp>
```

Inheritance diagram for GUIVisualizer::Node:



## Public Types

- enum `State` {
 `Default` , `Active` , `ActiveBlue` , `ActiveGreen` ,
 `ActiveRed` , `Iterated` , `Hide` , `StateCount` }
- The state of the node.*
- enum `Shape` { `Circle` , `Square` , `ShapeCount` }

- `typedef std::shared_ptr< Component > Ptr`  
*The shared pointer to the component.*

## Public Member Functions

- `void SetShape (Shape shape)`  
*Set the shape of the node.*
- `Shape GetShape () const`  
*Get the shape of the node.*
- `Node (int value, FontHolder *fonts)`  
*Construct a new `Node` object.*
- `Node ()`  
*Construct a new `Node` object.*
- `~Node ()`  
*Destroy the `Node` object.*
- `bool isSelectable () const`  
*Return true if the node is selectable.*
- `void Draw (Vector2 base=(Vector2){0, 0}, float t=1.0f)`  
*Draw the node.*
- `void SetActive (bool active)`  
*Set the active state of the node.*
- `bool IsActive ()`  
*Return true if the node is active.*
- `void SetValue (int value)`  
*Set the value of the node.*
- `int GetValue () const`  
*Get the value of the node.*
- `void SetLabel (std::string label)`  
*Set the label of the node.*
- `void ClearLabel ()`  
*Get the label of the node.*
- `void AnimationOnNode (bool animate)`
- `void SetRadius (float radius)`
- `void SetValueFontSize (int fontSize)`
- `void SetLabelFontSize (int fontSize)`
- `void SetNodeState (State state)`  
*Set the color of the node.*
- `State GetNodeState () const`  
*Get the color of the node.*
- `void SetReachable (bool reachable)`
- `bool GetReachable () const`
- `virtual void Draw (Vector2 base=(Vector2){0, 0})`  
*Draw the component.*
- `bool isSelected () const`  
*Check if the component is selected.*
- `virtual void select ()`  
*Select the component.*
- `virtual void deselect ()`  
*Deselect the component.*

- virtual void [ToggleVisible \(\)](#)  
*Toggle the visibility of the component.*
- virtual void [SetVisible \(bool visible\)](#)  
*Set the visibility of the component.*
- virtual bool [GetVisible \(\)](#)  
*Get the visibility of the component.*
- void [SetPosition \(float x, float y\)](#)  
*Set the position of the component.*
- void [SetPosition \(Vector2 position\)](#)  
*Set the position of the component.*
- Vector2 [GetPosition \(\)](#)  
*Get the position of the component.*
- virtual Vector2 [GetSize \(\)](#)  
*Get the size of the component.*

## Protected Member Functions

- virtual void [UpdateMouseCursorWhenHover \(std::map< std::string, Rectangle > bounds, bool hover, bool noHover\)](#)  
*Update the mouse cursor when hovering.*
- virtual void [UpdateMouseCursorWhenHover \(Rectangle bound, bool hover, bool noHover\)](#)  
*Update the mouse cursor when hovering.*
- virtual bool [GetHoverStatus \(std::map< std::string, Rectangle > bounds, bool hover, bool noHover\)](#)  
*Get the hover status of the component.*
- virtual bool [GetHoverStatus \(Rectangle bound, bool hover, bool noHover\)](#)  
*Get the hover status of the component.*

## Protected Attributes

- Vector2 [mPosition](#)  
*The position of the component.*
- bool [mVisible](#)  
*The visibility of the component.*

## Private Member Functions

- void [DrawLabel \(Vector2 base=\(Vector2\){0, 0}\)](#)
- void [DrawNode \(Vector2 base=\(Vector2\){0, 0}, float t=1.0f\)](#)  
*Draw the node.*
- Color [GetOutlineColor \(float t=1.0f\)](#)
- Color [GetBackgroundColor \(float t=1.0f\)](#)
- Color [GetTextColor \(float t=1.0f\)](#)
- void [AddColor \(\)](#)

## Private Attributes

- `Shape mShape = Shape::Circle`  
*The shape of the node, by default it is a circle.*
- `std::map< State, std::pair< ColorTheme::ID, ColorTheme::ID > > mOutlineColor`
- `std::map< State, std::pair< ColorTheme::ID, ColorTheme::ID > > mBackgroundColor`
- `std::map< State, std::pair< ColorTheme::ID, ColorTheme::ID > > mTextColor`
- `int mValue`
- `float mRadius`
- `float valueFontSize`
- `float labelFontSize`
- `std::string mLabel`
- `State mNodeState`
- `bool mReachable`
- `bool animateNode`
- `bool mActive`
- `FontHolder * fonts`
- `Color mDefaultColor`
- `Color mActiveColor`
- `Color mBorderColor`
- `bool mIsSelected`

*The selected status of the component.*

### 7.40.1 Detailed Description

The node class that is used to represent a node in the visualization.

### 7.40.2 Member Typedef Documentation

#### 7.40.2.1 Ptr

```
typedef std::shared_ptr< Component > GUI::Component::Ptr [inherited]
```

The shared pointer to the component.

The `GUI` component is commonly used by multiple components, so a shared pointer is used.

See also

`std::shared_ptr`

### 7.40.3 Member Enumeration Documentation

#### 7.40.3.1 Shape

```
enum GIVisualizer::Node::Shape
```

The shape of the node.

The shape of the node is used to determine the shape of the node.

**Enumerator**

Circle	
Square	
ShapeCount	

**7.40.3.2 State**

```
enum GUIVisualizer::Node::State
```

The state of the node.

The state of the node is used to determine the color of the node.

**Enumerator**

Default	
Active	
ActiveBlue	
ActiveGreen	
ActiveRed	
Iterated	
Hide	
StateCount	

**7.40.4 Constructor & Destructor Documentation****7.40.4.1 Node() [1/2]**

```
GUIVisualizer::Node::Node (
    int value,
    FontHolder * fonts )
```

Construct a new [Node](#) object.

**Parameters**

<i>value</i>	The value of the node.
<i>fonts</i>	The fonts of the application.

#### 7.40.4.2 Node() [2/2]

```
GUIVisualizer::Node::Node ( )
```

Construct a new [Node](#) object.

##### Parameters

<i>value</i>	The value of the node.
<i>label</i>	The label of the node.
<i>fonts</i>	The fonts of the application.

#### 7.40.4.3 ~Node()

```
GUIVisualizer::Node::~Node ( )
```

Destroy the [Node](#) object.

### 7.40.5 Member Function Documentation

#### 7.40.5.1 AddColor()

```
void GUIVisualizer::Node::AddColor ( ) [private]
```

#### 7.40.5.2 AnimationOnNode()

```
void GUIVisualizer::Node::AnimationOnNode (
    bool animate )
```

#### 7.40.5.3 ClearLabel()

```
void GUIVisualizer::Node::ClearLabel ( )
```

Get the label of the node.

#### 7.40.5.4 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

#### 7.40.5.5 Draw() [1/2]

```
void GUI::Component::Draw (
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Draw the component.

This function draws the component.

##### Parameters

<i>base</i>	The base position of the component.
-------------	-------------------------------------

Reimplemented in [GUIComponent::Button](#), [GUIComponent::Card](#), [GUIComponent::CodeHighlighter](#), [GUIComponent::InputField](#), [GUIComponent::NavigationBar](#), [GUIComponent::OperationList](#), and [GUI::Container](#).

#### 7.40.5.6 Draw() [2/2]

```
void GUIVisualizer::Node::Draw (
    Vector2 base = (Vector2){0, 0},
    float t = 1.0f )
```

Draw the node.

##### Parameters

<i>base</i>	The base position of the node.
<i>t</i>	The progress of the animation.

#### 7.40.5.7 DrawLabel()

```
void GUIVisualizer::Node::DrawLabel (
    Vector2 base = (Vector2){0, 0} ) [private]
```

#### 7.40.5.8 DrawNode()

```
void GUIVisualizer::Node::DrawNode (
    Vector2 base = (Vector2){0, 0},
    float t = 1.0f ) [private]
```

Draw the node.

##### Parameters

<i>base</i>	The base position of the node.
<i>t</i>	The progress of the animation.

#### 7.40.5.9 GetBackgroundColor()

```
Color GUIVisualizer::Node::GetBackgroundColor (
    float t = 1.0f ) [private]
```

#### 7.40.5.10 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.40.5.11 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.40.5.12 GetNodeState()**

```
GUIVisualizer::Node::State GUIVisualizer::Node::GetNodeState ( ) const
```

Get the color of the node.

**7.40.5.13 GetOutlineColor()**

```
Color GUIVisualizer::Node::GetOutlineColor (   
    float t = 1.0f ) [private]
```

**7.40.5.14 GetPosition()**

```
Vector2 GUI::Component::GetPosition ( ) [inherited]
```

Get the position of the component.

This function gets the position of the component.

**Returns**

The position of the component.

**7.40.5.15 GetReachable()**

```
bool GUIVisualizer::Node::GetReachable ( ) const
```

**7.40.5.16 GetShape()**

```
GUIVisualizer::Node::Shape GUIVisualizer::Node::GetShape ( ) const
```

Get the shape of the node.

#### 7.40.5.17 GetSize()

```
Vector2 GUI::Component::GetSize ( ) [virtual], [inherited]
```

Get the size of the component.

This function gets the size of the component.

##### Returns

The size of the component.

Reimplemented in [GUIComponent::Button](#), [GUIComponent::InputField](#), [GUIComponent::OperationList](#), [GUIComponent::OptionInputField](#) and [GUI::Container](#).

#### 7.40.5.18 GetTextColor()

```
Color GUVISUALIZER::Node::GetTextColor (
    float t = 1.0f ) [private]
```

#### 7.40.5.19 GetValue()

```
int GUVISUALIZER::Node::GetValue ( ) const
```

Get the value of the node.

#### 7.40.5.20 GetVisible()

```
bool GUI::Component::GetVisible ( ) [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

##### Returns

The visibility of the component.

#### 7.40.5.21 IsActive()

```
bool GUIVisualizer::Node::IsActive ( )
```

Return true if the node is active.

#### 7.40.5.22 IsSelectable()

```
bool GUIVisualizer::Node::IsSelectable ( ) const [virtual]
```

Return true if the node is selectable.

##### Note

The node is unselectable.

Implements [GUI::Component](#).

#### 7.40.5.23 IsSelected()

```
bool GUI::Component::IsSelected ( ) const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.40.5.24 select()

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

#### 7.40.5.25 SetActive()

```
void GUIVisualizer::Node::SetActive ( bool active )
```

Set the active state of the node.

**Parameters**

<i>active</i>	The active state of the node.
---------------	-------------------------------

**7.40.5.26 SetLabel()**

```
void GUIVisualizer::Node::SetLabel (
    std::string label )
```

Set the label of the node.

**Parameters**

<i>label</i>	The label of the node.
--------------	------------------------

**7.40.5.27 SetLabelFontSize()**

```
void GUIVisualizer::Node::SetLabelFontSize (
    int fontSize )
```

**7.40.5.28 SetNodeState()**

```
void GUIVisualizer::Node::SetNodeState (
    State state )
```

Set the color of the node.

**Parameters**

<i>color</i>	The color of the node.
--------------	------------------------

**7.40.5.29 SetPosition() [1/2]**

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

**7.40.5.30 SetPosition() [2/2]**

```
void GUI::Component::SetPosition (
    Vector2 position ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>position</i>	The position of the component.
-----------------	--------------------------------

**7.40.5.31 SetRadius()**

```
void GUIVisualizer::Node::SetRadius (
    float radius )
```

**7.40.5.32 SetReachable()**

```
void GUIVisualizer::Node::SetReachable (
    bool reachable )
```

**7.40.5.33 SetShape()**

```
void GUIVisualizer::Node::SetShape (
    Shape shape )
```

Set the shape of the node.

**7.40.5.34 SetValue()**

```
void GUIVisualizer::Node::SetValue (
    int value )
```

Set the value of the node.

**Parameters**

<i>value</i>	The value of the node.
--------------	------------------------

**7.40.5.35 SetValueFontSize()**

```
void GUIVisualizer::Node::SetValueFontSize (
    int fontSize )
```

**7.40.5.36 SetVisible()**

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

**Parameters**

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

**7.40.5.37 ToggleVisible()**

```
void GUI::Component::ToggleVisible () [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

**7.40.5.38 UpdateMouseCursorWhenHover() [1/2]**

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

**7.40.5.39 UpdateMouseCursorWhenHover() [2/2]**

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.40.6 Member Data Documentation****7.40.6.1 animateNode**

```
bool GUIVisualizer::Node::animateNode [private]
```

**7.40.6.2 fonts**

```
FontHolder* GUIVisualizer::Node::fonts [private]
```

**7.40.6.3 labelFontSize**

```
float GUIVisualizer::Node::labelFontSize [private]
```

#### 7.40.6.4 mActive

```
bool GUIVisualizer::Node::mActive [private]
```

#### 7.40.6.5 mActiveColor

```
Color GUIVisualizer::Node::mActiveColor [private]
```

#### 7.40.6.6 mBackgroundColor

```
std::map< State, std::pair< ColorTheme::ID, ColorTheme::ID > > GUIVisualizer::Node::mBackground->  
Color [private]
```

#### 7.40.6.7 mBorderColor

```
Color GUIVisualizer::Node::mBorderColor [private]
```

#### 7.40.6.8 mDefaultColor

```
Color GUIVisualizer::Node::mDefaultColor [private]
```

#### 7.40.6.9 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

#### 7.40.6.10 mLabel

```
std::string GUIVisualizer::Node::mLabel [private]
```

**7.40.6.11 mNodeState**

```
State GUIVisualizer::Node::mNodeState [private]
```

**7.40.6.12 mOutlineColor**

```
std::map< State, std::pair< ColorTheme::ID, ColorTheme::ID > > GUIVisualizer::Node::mOutlineColor [private]
```

**7.40.6.13 mPosition**

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

**7.40.6.14 mRadius**

```
float GUIVisualizer::Node::mRadius [private]
```

**7.40.6.15 mReachable**

```
bool GUIVisualizer::Node::mReachable [private]
```

**7.40.6.16 mShape**

```
Shape GUIVisualizer::Node::mShape = Shape::Circle [private]
```

The shape of the node, by default it is a circle.

**7.40.6.17 mTextColor**

```
std::map< State, std::pair< ColorTheme::ID, ColorTheme::ID > > GUIVisualizer::Node::mTextColor [private]
```

#### 7.40.6.18 mValue

```
int GUIVisualizer::Node::mValue [private]
```

#### 7.40.6.19 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

#### 7.40.6.20 valueFontSize

```
float GUIVisualizer::Node::valueFontSize [private]
```

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

- [src/Components/Visualization/Node.hpp](#)
- [src/Components/Visualization/Node.cpp](#)

## 7.41 NonCopyable< T > Class Template Reference

The self-explanatory non-copyable class.

```
#include <NonCopyable.hpp>
```

### Public Member Functions

- [NonCopyable](#) (const [NonCopyable](#) &)=delete  
*The copy constructor is deleted.*
- [T & operator=](#) (const T &)=delete  
*The assignment operator is deleted.*

### Protected Member Functions

- [NonCopyable](#) ()=default  
*Construct a new [NonCopyable](#) object.*
- [~NonCopyable](#) ()=default  
*Destroy the [NonCopyable](#) object.*

#### 7.41.1 Detailed Description

```
template<class T>
class NonCopyable< T >
```

The self-explanatory non-copyable class.

This class is used to make a class non-copyable.

## Template Parameters

<i>T</i>	The class that is going to be made non-copyable.
----------	--

## 7.41.2 Constructor & Destructor Documentation

### 7.41.2.1 NonCopyable() [1/2]

```
template<class T >
NonCopyable< T >::NonCopyable (
    const NonCopyable< T > & )  [delete]
```

The copy constructor is deleted.

The copy constructor is deleted to make the class non-copyable.

### 7.41.2.2 NonCopyable() [2/2]

```
template<class T >
NonCopyable< T >::NonCopyable ( )  [protected], [default]
```

Construct a new [NonCopyable](#) object.

This constructor is used to construct a new [NonCopyable](#) object.

### 7.41.2.3 ~NonCopyable()

```
template<class T >
NonCopyable< T >::~NonCopyable ( )  [protected], [default]
```

Destroy the [NonCopyable](#) object.

This destructor is used to destroy the [NonCopyable](#) object.

## 7.41.3 Member Function Documentation

#### 7.41.3.1 operator=()

```
template<class T >
T & NonCopyable< T >::operator= (
    const T & ) [delete]
```

The assignment operator is deleted.

The assignment operator is deleted to make the class non-copyable.

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

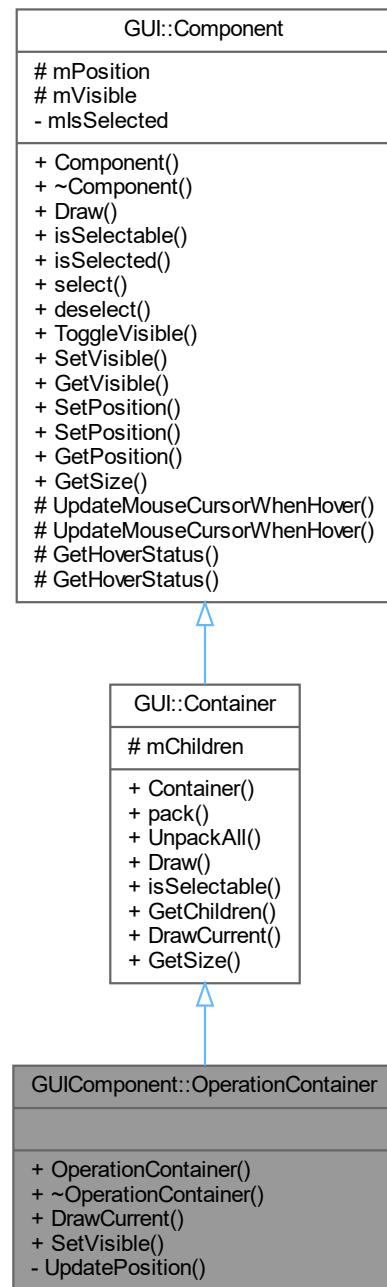
- src/[NonCopyable.hpp](#)

## 7.42 GUIComponent::OperationContainer Class Reference

The operation container class that is used to represent an operation container in the [GUI](#).

```
#include <OperationContainer.hpp>
```

Inheritance diagram for GUIComponent::OperationContainer:



## Public Types

- `typedef std::shared_ptr<OperationContainer> Ptr`

## Public Member Functions

- `OperationContainer ()`

- `~OperationContainer ()`
- void `DrawCurrent` (Vector2 base=(Vector2){0, 0})
- void `SetVisible` (bool visible)
 

*Set the visibility of the component.*
- void `pack` (Component::Ptr component)
 

*Pack a component into the container.*
- void `UnpackAll` ()
 

*Unpack all components from the container.*
- virtual void `Draw` (Vector2 base=(Vector2){0, 0})
 

*Draw the container.*
- virtual bool `isSelectable` () const
 

*Check if the container is selectable.*
- `Core::Deque< Component::Ptr > GetChildren ()`

*Get the pack components of the container.*
- virtual Vector2 `GetSize` ()
 

*Get the size of the container.*
- bool `isSelected` () const
 

*Check if the component is selected.*
- virtual void `select` ()
 

*Select the component.*
- virtual void `deselect` ()
 

*Deselect the component.*
- virtual void `ToggleVisible` ()
 

*Toggle the visibility of the component.*
- virtual bool `GetVisible` ()
 

*Get the visibility of the component.*
- void `SetPosition` (float x, float y)
 

*Set the position of the component.*
- void `SetPosition` (Vector2 position)
 

*Set the position of the component.*
- Vector2 `GetPosition` ()
 

*Get the position of the component.*

## Protected Member Functions

- virtual void `UpdateMouseCursorWhenHover` (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)
 

*Update the mouse cursor when hovering.*
- virtual void `UpdateMouseCursorWhenHover` (Rectangle bound, bool hover, bool noHover)
 

*Update the mouse cursor when hovering.*
- virtual bool `GetHoverStatus` (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)
 

*Get the hover status of the component.*
- virtual bool `GetHoverStatus` (Rectangle bound, bool hover, bool noHover)
 

*Get the hover status of the component.*

## Protected Attributes

- `Core::Deque< Component::Ptr > mChildren`
- Vector2 `mPosition`

*The position of the component.*
- bool `mVisible`

*The visibility of the component.*

## Private Member Functions

- void [UpdatePosition \(\)](#)

## Private Attributes

- bool [mIsSelected](#)

*The selected status of the component.*

### 7.42.1 Detailed Description

The operation container class that is used to represent an operation container in the [GUI](#).

The operation container will be used to store a list of operations that are of the same type. (e.g. Insert head, Insert after tail, etc.)

### 7.42.2 Member Typedef Documentation

#### 7.42.2.1 Ptr

```
typedef std::shared_ptr< OperationContainer > GUIComponent::OperationContainer::Ptr
```

### 7.42.3 Constructor & Destructor Documentation

#### 7.42.3.1 OperationContainer()

```
GUIComponent::OperationContainer::OperationContainer ( )
```

#### 7.42.3.2 ~OperationContainer()

```
GUIComponent::OperationContainer::~OperationContainer ( )
```

### 7.42.4 Member Function Documentation

#### 7.42.4.1 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

#### 7.42.4.2 Draw()

```
void GUI::Container::Draw (
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Draw the container.

This function draws the container.

##### Parameters

<i>base</i>	The base position of the container.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#).

#### 7.42.4.3 DrawCurrent()

```
void GUIComponent::OperationContainer::DrawCurrent (
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Reimplemented from [GUI::Container](#).

#### 7.42.4.4 GetChildren()

```
Core::Deque< GUI::Component::Ptr > GUI::Container::GetChildren ( ) [inherited]
```

Get the pack components of the container.

This function returns the all of the pack components of the container.

##### Returns

The children of the container.

#### 7.42.4.5 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.42.4.6 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

#### 7.42.4.7 GetPosition()

```
Vector2 GUI::Component::GetPosition () [inherited]
```

Get the position of the component.

This function gets the position of the component.

##### Returns

The position of the component.

#### 7.42.4.8 GetSize()

```
Vector2 GUI::Container::GetSize ( ) [virtual], [inherited]
```

Get the size of the container.

This function returns the size of the container.

##### Returns

The size of the container.

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#), and [GUIComponent::OptionInputField](#).

#### 7.42.4.9 GetVisible()

```
bool GUI::Component::GetVisible ( ) [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

##### Returns

The visibility of the component.

#### 7.42.4.10 isSelectable()

```
bool GUI::Container::isSelectable ( ) const [virtual], [inherited]
```

Check if the container is selectable.

**Deprecated** This function is deprecated.

This function checks if the container is selectable.

##### Returns

True if the container is selectable.

Implements [GUI::Component](#).

Reimplemented in [GUIVisualizer::CircularLinkedList](#), [GUIVisualizer::DoublyLinkedList](#), [GUIVisualizer::DynamicArray](#), [GUIVisualizer::LinkedList](#), and [GUIVisualizer::SinglyLinkedList](#).

#### 7.42.4.11 isSelected()

```
bool GUI::Component::isSelected ( ) const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.42.4.12 pack()

```
void GUI::Container::pack (
    Component::Ptr component ) [inherited]
```

Pack a component into the container.

This function packs a component into the container.

##### Parameters

component	The component to pack.
-----------	------------------------

#### 7.42.4.13 select()

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

#### 7.42.4.14 SetPosition() [1/2]

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

**7.42.4.15 SetPosition() [2/2]**

```
void GUI::Component::SetPosition (
    Vector2 position ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>position</i>	The position of the component.
-----------------	--------------------------------

**7.42.4.16 SetVisible()**

```
void GUIComponent::OperationContainer::SetVisible (
    bool visible ) [virtual]
```

Set the visibility of the component.

This function sets the visibility of the component.

**Parameters**

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented from [GUI::Component](#).

**7.42.4.17 ToggleVisible()**

```
void GUI::Component::ToggleVisible ( ) [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.42.4.18 UnpackAll()

```
void GUI::Container::UnpackAll ( ) [inherited]
```

Unpack all components from the container.

This function unpacks all components from the container.

##### See also

[pack](#)

#### 7.42.4.19 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.42.4.20 UpdateMouseCursorWhenHover() [2/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

#### 7.42.4.21 **UpdatePosition()**

```
void GUIComponent::OperationContainer::UpdatePosition ( ) [private]
```

### 7.42.5 Member Data Documentation

#### 7.42.5.1 **mChildren**

```
Core::Deque< Component::Ptr > GUI::Container::mChildren [protected], [inherited]
```

#### 7.42.5.2 **mIsSelected**

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

#### 7.42.5.3 **mPosition**

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

#### 7.42.5.4 **mVisible**

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

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

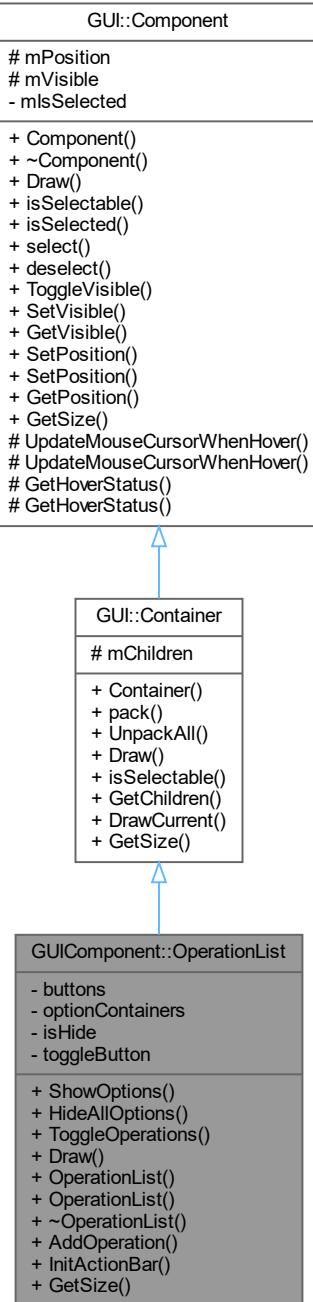
- src/Components/Common/[OperationContainer.hpp](#)
- src/Components/Common/[OperationContainer.cpp](#)

## 7.43 GUIComponent::OperationList Class Reference

The operation list class that is used to represent an operation list in the [GUI](#).

```
#include <OperationList.hpp>
```

Inheritance diagram for GUIComponent::OperationList:



## Public Types

- `typedef std::shared_ptr< Container > Ptr`  
*The shared pointer to the container.*

## Public Member Functions

- `void ShowOptions (std::size_t index)`
- `void HideAllOptions ()`
- `void ToggleOperations ()`
- `void Draw (Vector2 base=(Vector2){0, 0})`  
*Draw the container.*
- `OperationList ()`
- `OperationList (FontHolder *fonts)`
- `~OperationList ()`
- `void AddOperation (GUIComponent::Button::Ptr action, GUI::Container::Ptr optionContainer)`
- `void InitActionBar ()`
- `Vector2 GetSize ()`  
*Get the size of the container.*
- `void pack (Component::Ptr component)`  
*Pack a component into the container.*
- `void UnpackAll ()`  
*Unpack all components from the container.*
- `virtual bool IsSelectable () const`  
*Check if the container is selectable.*
- `Core::Deque< Component::Ptr > GetChildren ()`  
*Get the pack components of the container.*
- `virtual void DrawCurrent (Vector2 base)`
- `bool isSelected () const`  
*Check if the component is selected.*
- `virtual void select ()`  
*Select the component.*
- `virtual void deselect ()`  
*Deselect the component.*
- `virtual void ToggleVisible ()`  
*Toggle the visibility of the component.*
- `virtual void SetVisible (bool visible)`  
*Set the visibility of the component.*
- `virtual bool GetVisible ()`  
*Get the visibility of the component.*
- `void SetPosition (float x, float y)`  
*Set the position of the component.*
- `void SetPosition (Vector2 position)`  
*Set the position of the component.*
- `Vector2 GetPosition ()`  
*Get the position of the component.*

## Protected Member Functions

- virtual void [UpdateMouseCursorWhenHover](#) (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)  
*Update the mouse cursor when hovering.*
- virtual void [UpdateMouseCursorWhenHover](#) (Rectangle bound, bool hover, bool noHover)  
*Update the mouse cursor when hovering.*
- virtual bool [GetHoverStatus](#) (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)  
*Get the hover status of the component.*
- virtual bool [GetHoverStatus](#) (Rectangle bound, bool hover, bool noHover)  
*Get the hover status of the component.*

## Protected Attributes

- [Core::Deque< Component::Ptr > mChildren](#)
- Vector2 [mPosition](#)  
*The position of the component.*
- bool [mVisible](#)  
*The visibility of the component.*

## Private Attributes

- [GUI::Container buttons](#)
- [GUI::Container optionContainers](#)
- bool [isHide](#)
- [GUIComponent::Button toggleButton](#)
- bool [mIsSelected](#)  
*The selected status of the component.*

### 7.43.1 Detailed Description

The operation list class that is used to represent an operation list in the [GUI](#).

The operation list will be used to display a list of operations that you can perform on the current data structure. (e.g Create/Initialize, Insert, Delete, etc.)

### 7.43.2 Member Typedef Documentation

#### 7.43.2.1 Ptr

```
typedef std::shared_ptr< Container > GUI::Container::Ptr [inherited]
```

The shared pointer to the container.

The [GUI](#) container is commonly used by multiple components, so a shared pointer is used.

#### See also

std::shared\_ptr

### 7.43.3 Constructor & Destructor Documentation

#### 7.43.3.1 OperationList() [1/2]

```
GUIComponent::OperationList::OperationList ( )
```

#### 7.43.3.2 OperationList() [2/2]

```
GUIComponent::OperationList::OperationList (
    FontHolder * fonts )
```

#### 7.43.3.3 ~OperationList()

```
GUIComponent::OperationList::~OperationList ( )
```

### 7.43.4 Member Function Documentation

#### 7.43.4.1 AddOperation()

```
void GUIComponent::OperationList::AddOperation (
    GUIComponent::Button::Ptr action,
    GUI::Container::Ptr optionContainer )
```

#### 7.43.4.2 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

#### 7.43.4.3 Draw()

```
void GUIComponent::OperationList::Draw (
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Draw the container.

This function draws the container.

**Parameters**

<i>base</i>	The base position of the container.
-------------	-------------------------------------

Reimplemented from [GUI::Container](#).

#### 7.43.4.4 DrawCurrent()

```
void GUI::Container::DrawCurrent (
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

#### 7.43.4.5 GetChildren()

```
Core::Deque< GUI::Component::Ptr > GUI::Container::GetChildren () [inherited]
```

Get the pack components of the container.

This function returns the all of the pack components of the container.

**Returns**

The children of the container.

#### 7.43.4.6 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.43.4.7 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

#### 7.43.4.8 GetPosition()

```
Vector2 GUI::Component::GetPosition () [inherited]
```

Get the position of the component.

This function gets the position of the component.

##### Returns

The position of the component.

#### 7.43.4.9 GetSize()

```
Vector2 GUIComponent::OperationList::GetSize () [virtual]
```

Get the size of the container.

This function returns the size of the container.

##### Returns

The size of the container.

Reimplemented from [GUI::Container](#).

#### 7.43.4.10 GetVisible()

```
bool GUI::Component::GetVisible ( ) [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

##### Returns

The visibility of the component.

#### 7.43.4.11 HideAllOptions()

```
void GUIComponent::OperationList::HideAllOptions ( )
```

#### 7.43.4.12 InitActionBar()

```
void GUIComponent::OperationList::InitActionBar ( )
```

#### 7.43.4.13 isSelectable()

```
bool GUI::Container::isSelectable ( ) const [virtual], [inherited]
```

Check if the container is selectable.

**Deprecated** This function is deprecated.

This function checks if the container is selectable.

##### Returns

True if the container is selectable.

Implements [GUI::Component](#).

Reimplemented in [GUIVisualizer::CircularLinkedList](#), [GUIVisualizer::DoublyLinkedList](#), [GUIVisualizer::DynamicArray](#), [GUIVisualizer::LinkedList](#), and [GUIVisualizer::SinglyLinkedList](#).

#### 7.43.4.14 isSelected()

```
bool GUI::Component::isSelected ( ) const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.43.4.15 pack()

```
void GUI::Container::pack (
    Component::Ptr component ) [inherited]
```

Pack a component into the container.

This function packs a component into the container.

##### Parameters

component	The component to pack.
-----------	------------------------

#### 7.43.4.16 select()

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

#### 7.43.4.17 SetPosition() [1/2]

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

**7.43.4.18 SetPosition() [2/2]**

```
void GUI::Component::SetPosition (
    Vector2 position ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>position</i>	The position of the component.
-----------------	--------------------------------

**7.43.4.19 SetVisible()**

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

**Parameters**

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

**7.43.4.20 ShowOptions()**

```
void GUIComponent::OperationList::ShowOptions (
    std::size_t index )
```

#### 7.43.4.21 **ToggleOperations()**

```
void GUIComponent::OperationList::ToggleOperations ( )
```

#### 7.43.4.22 **ToggleVisible()**

```
void GUI::Component::ToggleVisible ( ) [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.43.4.23 **UnpackAll()**

```
void GUI::Container::UnpackAll ( ) [inherited]
```

Unpack all components from the container.

This function unpacks all components from the container.

#### See also

[pack](#)

#### 7.43.4.24 **UpdateMouseCursorWhenHover() [1/2]**

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

#### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.43.4.25 **UpdateMouseCursorWhenHover() [2/2]**

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
```

```
    bool hover,  
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

#### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

## 7.43.5 Member Data Documentation

### 7.43.5.1 buttons

```
GUI::Container GUIComponent::OperationList::buttons [private]
```

### 7.43.5.2 isHide

```
bool GUIComponent::OperationList::isHide [private]
```

### 7.43.5.3 mChildren

```
Core::Deque< Component::Ptr > GUI::Container::mChildren [protected], [inherited]
```

### 7.43.5.4 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

#### 7.43.5.5 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

#### 7.43.5.6 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

#### 7.43.5.7 optionContainers

```
GUI::Container GUIComponent::OperationList::optionContainers [private]
```

#### 7.43.5.8 toggleButton

```
GUIComponent::Button GUIComponent::OperationList::toggleButton [private]
```

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

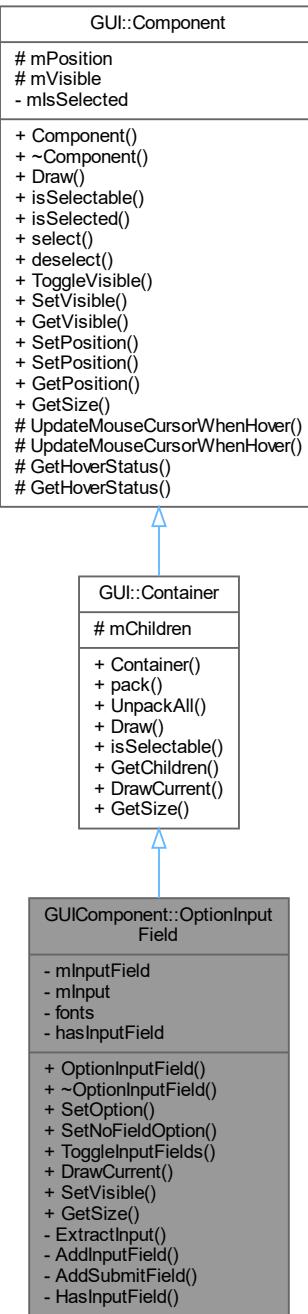
- src/Components/Common/[OperationList.hpp](#)
- src/Components/Common/[OperationList.cpp](#)

## 7.44 GUIComponent::OptionInputField Class Reference

The option input field class that is used to represent an option input field in the [GUI](#).

```
#include <OptionInputField.hpp>
```

Inheritance diagram for GUIComponent::OptionInputField:



## Public Types

- `typedef std::shared_ptr<OptionInputField> Ptr`

## Public Member Functions

- `OptionInputField (FontHolder *fonts)`

- `~OptionInputField ()`
- void `SetOption` (std::string content, Core::Deque< InputField::Ptr > fields, std::function< void(std::map< std::string, std::string >) > action)
- void `SetNoFieldOption` (std::string content, std::function< void() > action)
- void `ToggleInputFields ()`
- void `DrawCurrent` (Vector2 base=(Vector2){0, 0})
- void `SetVisible` (bool visible)
 

*Set the visibility of the component.*
- virtual Vector2 `GetSize` ()
 

*Get the size of the container.*
- void `pack` (Component::Ptr component)
 

*Pack a component into the container.*
- void `UnpackAll` ()
 

*Unpack all components from the container.*
- virtual void `Draw` (Vector2 base=(Vector2){0, 0})
 

*Draw the container.*
- virtual bool `isSelectable` () const
 

*Check if the container is selectable.*
- Core::Deque< Component::Ptr > `GetChildren` ()
 

*Get the pack components of the container.*
- bool `isSelected` () const
 

*Check if the component is selected.*
- virtual void `select` ()
 

*Select the component.*
- virtual void `deselect` ()
 

*Deselect the component.*
- virtual void `ToggleVisible` ()
 

*Toggle the visibility of the component.*
- virtual bool `GetVisible` ()
 

*Get the visibility of the component.*
- void `SetPosition` (float x, float y)
 

*Set the position of the component.*
- void `SetPosition` (Vector2 position)
 

*Set the position of the component.*
- Vector2 `GetPosition` ()
 

*Get the position of the component.*

## Protected Member Functions

- virtual void `UpdateMouseCursorWhenHover` (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)
 

*Update the mouse cursor when hovering.*
- virtual void `UpdateMouseCursorWhenHover` (Rectangle bound, bool hover, bool noHover)
 

*Update the mouse cursor when hovering.*
- virtual bool `GetHoverStatus` (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)
 

*Get the hover status of the component.*
- virtual bool `GetHoverStatus` (Rectangle bound, bool hover, bool noHover)
 

*Get the hover status of the component.*

## Protected Attributes

- `Core::Deque< Component::Ptr > mChildren`
- `Vector2 mPosition`  
*The position of the component.*
- `bool mVisible`  
*The visibility of the component.*

## Private Member Functions

- `std::map< std::string, std::string > ExtractInput ()`
- `void AddInputField (InputField::Ptr inputField)`
- `void AddSubmitField (std::function< void(std::map< std::string, std::string >) > action)`
- `bool HasInputField ()`

## Private Attributes

- `GUI::Container::Ptr mInputField`
- `std::map< std::string, std::string > mInput`
- `FontHolder * fonts`
- `bool hasInputField`
- `bool mIsSelected`

*The selected status of the component.*

### 7.44.1 Detailed Description

The option input field class that is used to represent an option input field in the [GUI](#).

The option input field is used to have a button that can be used to toggle the visibility of an input fields.

### 7.44.2 Member Typedef Documentation

#### 7.44.2.1 Ptr

```
typedef std::shared_ptr< OptionInputField > GUIComponent::OptionInputField::Ptr
```

### 7.44.3 Constructor & Destructor Documentation

#### 7.44.3.1 OptionInputField()

```
GUIComponent::OptionInputField::OptionInputField (
    FontHolder * fonts )
```

#### 7.44.3.2 ~OptionInputField()

```
GUIComponent::OptionInputField::~OptionInputField ( )
```

### 7.44.4 Member Function Documentation

#### 7.44.4.1 AddInputField()

```
void GUIComponent::OptionInputField::AddInputField (
    InputField::Ptr inputField ) [private]
```

#### 7.44.4.2 AddSubmitField()

```
void GUIComponent::OptionInputField::AddSubmitField (
    std::function< void(std::map< std::string, std::string >) > action ) [private]
```

#### 7.44.4.3 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

#### 7.44.4.4 Draw()

```
void GUI::Container::Draw (
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Draw the container.

This function draws the container.

**Parameters**

<code>base</code>	The base position of the container.
-------------------	-------------------------------------

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#).

#### 7.44.4.5 DrawCurrent()

```
void GUIComponent::OptionInputField::DrawCurrent (
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Reimplemented from [GUI::Container](#).

#### 7.44.4.6 ExtractInput()

```
std::map< std::string, std::string > GUIComponent::OptionInputField::ExtractInput () [private]
```

#### 7.44.4.7 GetChildren()

```
Core::Deque< GUI::Component::Ptr > GUI::Container::GetChildren () [inherited]
```

Get the pack components of the container.

This function returns the all of the pack components of the container.

**Returns**

The children of the container.

#### 7.44.4.8 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bound</i>	The bound of the component.
--------------	-----------------------------

**7.44.4.9 GetHoverStatus() [2/2]**

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

**Parameters**

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

**7.44.4.10 GetPosition()**

```
Vector2 GUI::Component::GetPosition () [inherited]
```

Get the position of the component.

This function gets the position of the component.

**Returns**

The position of the component.

**7.44.4.11 GetSize()**

```
Vector2 GUIComponent::OptionInputField::GetSize () [virtual]
```

Get the size of the container.

This function returns the size of the container.

**Returns**

The size of the container.

Reimplemented from [GUI::Container](#).

#### 7.44.4.12 GetVisible()

```
bool GUI::Component::GetVisible ( ) [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

##### Returns

The visibility of the component.

#### 7.44.4.13 HasInputField()

```
bool GUIComponent::OptionInputField::HasInputField ( ) [private]
```

#### 7.44.4.14 IsSelectable()

```
bool GUI::Container::IsSelectable ( ) const [virtual], [inherited]
```

Check if the container is selectable.

**Deprecated** This function is deprecated.

This function checks if the container is selectable.

##### Returns

True if the container is selectable.

Implements [GUI::Component](#).

Reimplemented in [GUIVisualizer::CircularLinkedList](#), [GUIVisualizer::DoublyLinkedList](#), [GUIVisualizer::DynamicArray](#), [GUIVisualizer::LinkedList](#), and [GUIVisualizer::SinglyLinkedList](#).

#### 7.44.4.15 IsSelected()

```
bool GUI::Component::IsSelected ( ) const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.44.4.16 pack()

```
void GUI::Container::pack ( Component::Ptr component ) [inherited]
```

Pack a component into the container.

This function packs a component into the container.

**Parameters**

<i>component</i>	The component to pack.
------------------	------------------------

**7.44.4.17 select()**

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

**7.44.4.18 SetNoFieldOption()**

```
void GUIComponent::OptionInputField::SetNoFieldOption (
    std::string content,
    std::function< void() > action )
```

**7.44.4.19 SetOption()**

```
void GUIComponent::OptionInputField::SetOption (
    std::string content,
    Core::Deque< InputField::Ptr > fields,
    std::function< void(std::map< std::string, std::string >) > action )
```

**7.44.4.20 SetPosition() [1/2]**

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

#### 7.44.4.21 SetPosition() [2/2]

```
void GUI::Component::SetPosition (
    Vector2 position )  [inherited]
```

Set the position of the component.

This function sets the position of the component.

##### Parameters

<i>position</i>	The position of the component.
-----------------	--------------------------------

#### 7.44.4.22 SetVisible()

```
void GUIComponent::OptionInputField::SetVisible (
    bool visible )  [virtual]
```

Set the visibility of the component.

This function sets the visibility of the component.

##### Parameters

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented from [GUI::Component](#).

#### 7.44.4.23 ToggleInputFields()

```
void GUIComponent::OptionInputField::ToggleInputFields ( )
```

#### 7.44.4.24 ToggleVisible()

```
void GUI::Component::ToggleVisible ( )  [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.44.4.25 UnpackAll()

```
void GUI::Container::UnpackAll ( ) [inherited]
```

Unpack all components from the container.

This function unpacks all components from the container.

##### See also

[pack](#)

#### 7.44.4.26 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.44.4.27 UpdateMouseCursorWhenHover() [2/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

#### 7.44.5 Member Data Documentation

#### 7.44.5.1 fonts

```
FontHolder* GUIComponent::OptionInputField::fonts [private]
```

#### 7.44.5.2 hasInputField

```
bool GUIComponent::OptionInputField::hasInputField [private]
```

#### 7.44.5.3 mChildren

```
Core::Deque< Component::Ptr > GUI::Container::mChildren [protected], [inherited]
```

#### 7.44.5.4 mInput

```
std::map< std::string, std::string > GUIComponent::OptionInputField::mInput [private]
```

#### 7.44.5.5 mInputField

```
GUI::Container::Ptr GUIComponent::OptionInputField::mInputField [private]
```

#### 7.44.5.6 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

#### 7.44.5.7 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

#### 7.44.5.8 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

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

- src/Components/Common/[OptionInputField.hpp](#)
- src/Components/Common/[OptionInputField.cpp](#)

## 7.45 State::StateStack::PendingChange Struct Reference

The pending changes that are applied to the state stack.

### Public Member Functions

- [PendingChange \(Action action, States::ID stateID=States::None\)](#)
- [PendingChange \(\)](#)

### Public Attributes

- [Action action](#)
- [States::ID stateID](#)

#### 7.45.1 Detailed Description

The pending changes that are applied to the state stack.

The pending changes can be one of the following:

- Push: Add a new state to the top of the stack (or equivalent to switching to a new state/scene)
- Pop: Remove the top state from the stack (or equivalent to returning to the previous state/scene, or exiting the application if the stack is empty)
- Clear: Remove all the states from the stack (or equivalent to clearing the stack)

See also

[Action](#)

[StateStack::ApplyPendingChanges](#)

#### 7.45.2 Constructor & Destructor Documentation

### 7.45.2.1 PendingChange() [1/2]

```
State::StateStack::PendingChange::PendingChange (
    Action action,
    States::ID stateID = States::None ) [explicit]
```

### 7.45.2.2 PendingChange() [2/2]

```
State::StateStack::PendingChange::PendingChange ( )
```

## 7.45.3 Member Data Documentation

### 7.45.3.1 action

```
Action State::StateStack::PendingChange::action
```

### 7.45.3.2 stateID

```
States::ID State::StateStack::PendingChange::stateID
```

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

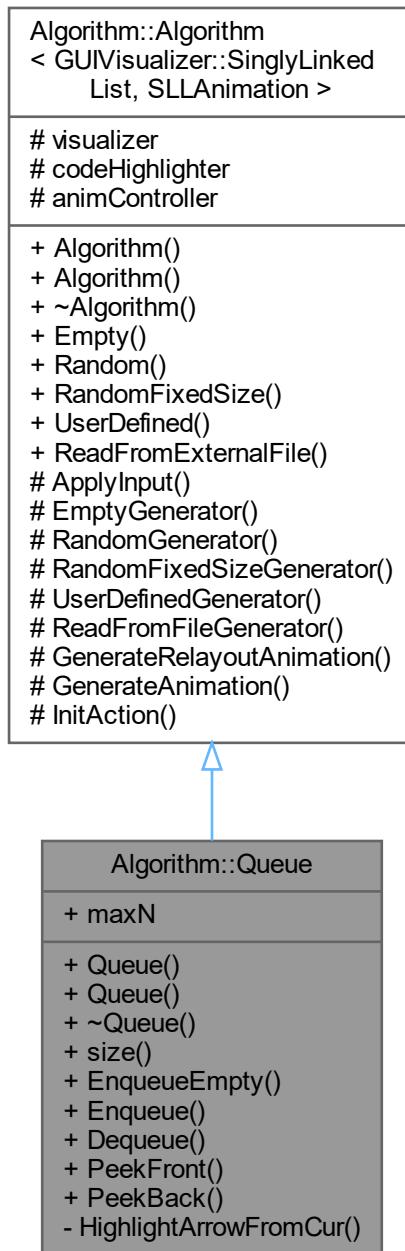
- src/[StateStack.hpp](#)
- src/[StateStack.cpp](#)

## 7.46 Algorithm::Queue Class Reference

The algorithm class that is used to generate step-by-step instructions for the visualization of the queue.

```
#include <Queue.hpp>
```

Inheritance diagram for Algorithm::Queue:



## Public Member Functions

- `Queue (GUIComponent::CodeHighlighter::Ptr _codeHighlighter, SLLAnimationController::Ptr animController, FontHolder *fonts)`  
*The constructor of the queue algorithm.*
- `Queue ()`  
*The destructor of the queue algorithm.*

- `~Queue ()`  
*Return the size of the queue.*
- `std::size_t size () const`  
*Return the size of the queue.*
- `void EnqueueEmpty (int value)`  
*Insert a new element when the queue is empty.*
- `void Enqueue (int value)`  
*Insert a new element at the back of the queue (after the tail of the singly linked list).*
- `void Dequeue ()`  
*Remove the element at the front of the queue (the head of the singly linked list).*
- `void PeekFront ()`  
*Return the value of the element at the front of the queue (the head of the singly linked list).*
- `void PeekBack ()`  
*Return the value of the element at the back of the queue (the tail of the singly linked list).*
- `virtual void Empty ()`  
*Initialize an empty data structure, and generate animation.*
- `virtual void Random ()`  
*Initialize a data structure with random values input, and then generate animation.*
- `virtual void RandomFixedSize (int N)`  
*Initialize a fixed size data structure with random values input, and then generate animation.*
- `virtual void UserDefined (std::string input)`  
*Initialize a data structure with input from user, and then generate animation.*
- `virtual void ReadFromExternalFile (std::string path)`  
*Initialize a data structure with input from external file (which is used to store the input), and then generate animation.*

## Static Public Attributes

- `static constexpr int maxN = 16`  
*The maximum number of elements that the queue can hold.*

## Protected Member Functions

- `virtual void ApplyInput (std::vector< int > input, std::size_t nMaxSize=10)`  
*Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an std::vector< int >) to the data structure, used by other initialization functions.*
- `std::vector< int > EmptyGenerator ()`  
*Generate an empty input, this is used to generate the input for `Empty ()`*
- `std::vector< int > RandomGenerator ()`  
*Generate a random input, this is used to generate the input for `Random ()`*
- `std::vector< int > RandomFixedSizeGenerator (int nSize)`  
*Generate a random input with fixed size, this is used to generate the input for `RandomFixedSize ()`*
- `std::vector< int > UserDefinedGenerator (std::string input)`  
*Generate an input from user, this is used to generate the input for `UserDefined ()`*
- `std::vector< int > ReadFromFileGenerator (std::string inputFile)`  
*Parse the input from external file, this is used to generate the input for `ReadFromExternalFile ()`*
- `virtual void GenerateRelayoutAnimation (Vector2 newPosition)`  
*Generate the relayout animation (which reposition the data structure to a new position on the screen)*
- `virtual SLLAnimation GenerateAnimation (float duration, int highlightLine, std::string actionDescription)`  
*Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)*
- `virtual void InitAction (std::vector< std::string > code)`  
*Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.*

## Protected Attributes

- `GUIVisualizer::SinglyLinkedList visualizer`  
*Visualizer for the algorithm (which is used to draw animation generated by the algorithm)*
- `GUICOMPONENT::CodeHighlighter::Ptr codeHighlighter`  
*Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)*
- `Animation::AnimationController< SLLAnimation >::Ptr animController`  
*Animation controller for the algorithm (which is used to control the animation generated by the algorithm)*

## Private Member Functions

- `std::function< GUIVisualizer::SinglyLinkedList(GUIVisualizer::SinglyLinkedList, float, Vector2) > HighlightArrowFromCur`  
`(int index, bool drawVisualizer=true, bool reverse=false)`  
*Reset the visualizer of the singly linked list algorithm.*

### 7.46.1 Detailed Description

The algorithm class that is used to generate step-by-step instructions for the visualization of the queue.

### 7.46.2 Constructor & Destructor Documentation

#### 7.46.2.1 Queue() [1/2]

```
Algorithm::Queue::Queue (
    GUICOMPONENT::CodeHighlighter::Ptr _codeHighlighter,
    SLLAnimationController::Ptr animController,
    FontHolder * fonts )
```

The constructor of the queue algorithm.

#### Parameters

<code>codeHighlighter</code>	The code highlighter of the queue algorithm.
<code>animController</code>	The animation controller of the queue algorithm.
<code>fonts</code>	The fonts of the queue algorithm.

#### 7.46.2.2 Queue() [2/2]

```
Algorithm::Queue::Queue ( )
```

The destructor of the queue algorithm.

**Note**

This destructor is not virtual because this class is not designed to be inherited.

**7.46.2.3 ~Queue()**

```
Algorithm::Queue::~Queue ( )
```

Return the size of the queue.

**Returns**

The size of the queue.

**7.46.3 Member Function Documentation****7.46.3.1 ApplyInput()**

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::ApplyInput (
    std::vector< int > input,
    std::size_t nMaxSize = 10 ) [protected], [virtual], [inherited]
```

Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an `std::vector< int >`) to the data structure, used by other initialization functions.

**Parameters**

<code>input</code>	the input (in <code>std::vector&lt; int &gt;</code> ) to be applied to the data structure
<code>nMaxSize</code>	the maximum size of the data structure, if the input size is larger than <code>nMaxSize</code> , the input will be truncated to <code>nMaxSize</code>

**7.46.3.2 Dequeue()**

```
void Algorithm::Queue::Dequeue ( )
```

Remove the element at the front of the queue (the head of the singly linked list).

**7.46.3.3 Empty()**

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::Empty [virtual],
[inherited]
```

Initialize an empty data structure, and generate animation.

#### 7.46.3.4 EmptyGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::←
EmptyGenerator [protected], [inherited]
```

Generate an empty input, this is used to generate the input for [Empty \(\)](#)

##### Returns

`std::vector< int >` the empty input

#### 7.46.3.5 Enqueue()

```
void Algorithm::Queue::Enqueue (
    int value )
```

Insert a new element at the back of the queue (after the tail of the singly linked list).

##### Parameters

<code>value</code>	The value of the new element.
--------------------	-------------------------------

#### 7.46.3.6 EnqueueEmpty()

```
void Algorithm::Queue::EnqueueEmpty (
    int value )
```

Insert a new element when the queue is empty.

##### Parameters

<code>value</code>	The value of the new element.
--------------------	-------------------------------

#### 7.46.3.7 GenerateAnimation()

```
SLLAnimation Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::Generate←
Animation (
    float duration,
    int highlightLine,
    std::string actionDescription ) [protected], [virtual], [inherited]
```

Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)

**Parameters**

<i>duration</i>	the duration of the animation
<i>highlightLine</i>	the line to be highlighted in the code highlighter (if the line is -1, then no line will be highlighted)
<i>actionDescription</i>	the description of the action (which is used to display in the animation)

**Returns**

AnimationState the empty animation state (which currently only contains the metadata of the animation)

**7.46.3.8 GenerateRelayoutAnimation()**

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::GenerateRelayout<->
Animation (
    Vector2 newPosition ) [inline], [protected], [virtual], [inherited]
```

Generate the relayout animation (which reposition the data structure to a new position on the screen)

**Parameters**

<i>newPosition</i>	the new position of the data structure
--------------------	--

**7.46.3.9 HighlightArrowFromCur()**

```
std::function< GUIVisualizer::SinglyLinkedList (GUIVisualizer::SinglyLinkedList, float, Vector2)
> Algorithm::Queue::HighlightArrowFromCur (
    int index,
    bool drawVisualizer = true,
    bool reverse = false ) [private]
```

Reset the visualizer of the singly linked list algorithm.

**Note**

This function is used to reset all of the effects that are only used to visualize the algorithm (i.e highlight the elements that are currently iterated).

### 7.46.3.10 InitAction()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::InitAction ( std::vector< std::string > code ) [protected], [virtual], [inherited]
```

Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.

#### Note

This function is used to clear the animation controller and the code highlighter, and then initialize a new instruction

### 7.46.3.11 PeekBack()

```
void Algorithm::Queue::PeekBack ( )
```

Return the value of the element at the back of the queue (the tail of the singly linked list).

#### Returns

The value of the element at the back of the queue.

### 7.46.3.12 PeekFront()

```
void Algorithm::Queue::PeekFront ( )
```

Return the value of the element at the front of the queue (the head of the singly linked list).

#### Returns

The value of the element at the front of the queue.

### 7.46.3.13 Random()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::Random [virtual], [inherited]
```

Initialize a data structure with random values input, and then generate animation.

### 7.46.3.14 RandomFixedSize()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::RandomFixedSize ( int N ) [virtual], [inherited]
```

Initialize a fixed size data structure with random values input, and then generate animation.

**Parameters**

<i>N</i>	the size of the data structure
----------	--------------------------------

**7.46.3.15 RandomFixedSizeGenerator()**

```
std::vector< int > Algorithm::Algorithm< GIVVisualizer::SinglyLinkedList , SLLAnimation >::←
RandomFixedSizeGenerator (
    int nSize ) [protected], [inherited]
```

Generate a random input with fixed size, this is used to generate the input for [RandomFixedSize\(\)](#)

**Parameters**

<i>nSize</i>	the size of the input
--------------	-----------------------

**Returns**

```
std::vector< int > the random input with fixed size
```

**7.46.3.16 RandomGenerator()**

```
std::vector< int > Algorithm::Algorithm< GIVVisualizer::SinglyLinkedList , SLLAnimation >::←
RandomGenerator [protected], [inherited]
```

Generate a random input, this is used to generate the input for [Random\(\)](#)

**Returns**

```
std::vector< int > the random input
```

**7.46.3.17 ReadFromExternalFile()**

```
void Algorithm::Algorithm< GIVVisualizer::SinglyLinkedList , SLLAnimation >::ReadFromExternal←
File (
    std::string path ) [virtual], [inherited]
```

Initialize a data structure with input from external file (which is used to store the input), and then generate animation.

**Parameters**

<i>path</i>	the path to the external file
-------------	-------------------------------

### 7.46.3.18 ReadFromFileGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::←
ReadFromFileGenerator (
    std::string inputFile ) [protected], [inherited]
```

Parse the input from external file, this is used to generate the input for [ReadFromExternalFile\(\)](#)

#### Parameters

<i>inputFile</i>	the path to the external file
------------------	-------------------------------

#### Returns

```
std::vector< int > the input from external file
```

### 7.46.3.19 size()

```
std::size_t Algorithm::Queue::size ( ) const
```

Return the size of the queue.

#### Returns

The size of the queue.

### 7.46.3.20 UserDefined()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::UserDefined (
    std::string input ) [virtual], [inherited]
```

Initialize a data structure with input from user, and then generate animation.

#### Parameters

<i>input</i>	the input from user
--------------	---------------------

### 7.46.3.21 UserDefinedGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::←
```

```
UserDefinedGenerator (
    std::string input ) [protected], [inherited]
```

Generate an input from user, this is used to generate the input for [UserDefined\(\)](#)

#### Parameters

<i>input</i>	the input from user
--------------	---------------------

#### Returns

```
std::vector< int > the input from user
```

### 7.46.4 Member Data Documentation

#### 7.46.4.1 animController

```
Animation::AnimationController< SLLAnimation >::Ptr Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList ,
, SLLAnimation >::animController [protected], [inherited]
```

[Animation](#) controller for the algorithm (which is used to control the animation generated by the algorithm)

#### Note

This [Algorithm](#) class is mainly use the animation controller to add animation for the algorithm

#### 7.46.4.2 codeHighlighter

```
GUIComponent::CodeHighlighter::Ptr Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList ,
SLLAnimation >::codeHighlighter [protected], [inherited]
```

Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)

#### 7.46.4.3 maxN

```
constexpr int Algorithm::Queue::maxN = 16 [static], [constexpr]
```

The maximum number of elements that the queue can hold.

#### 7.46.4.4 visualizer

```
GUIVisualizer::SinglyLinkedList Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation
>::visualizer [protected], [inherited]
```

Visualizer for the algorithm (which is used to draw animation generated by the algorithm)

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

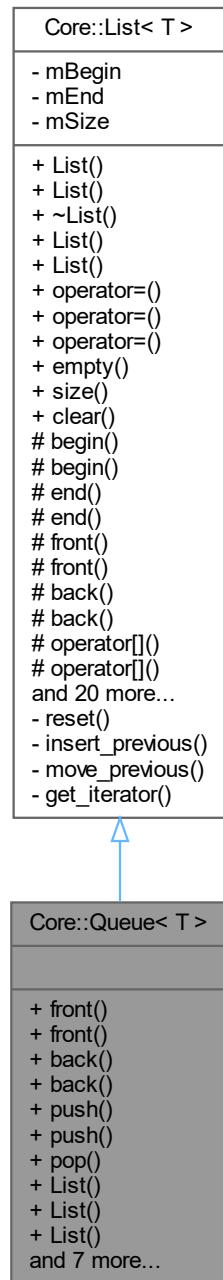
- src/Algorithms/LinkedList/Queue.hpp
- src/Algorithms/LinkedList/Queue.cpp

## 7.47 Core::Queue< T > Class Template Reference

The queue container.

```
#include <Queue.hpp>
```

Inheritance diagram for Core::Queue< T >:



## Public Member Functions

- `T & front ()`  
*Returns the reference to the front element of the queue.*
- `const T & front () const`  
*Returns the const reference to the front element of the queue.*
- `T & back ()`

- `const T & back () const`  
*Returns the reference to the back element of the queue.*
- `void push (const T &value)`  
*Returns the const reference to the back element of the queue.*
- `void push (T &&value)`  
*Inserts the element at the back of the queue.*
- `void pop ()`  
*Removes the front element of the queue.*
- `List ()`  
*Default constructor.*
- `List (std::initializer_list< T > list)`  
*Constructs the container with the contents of the initializer list.*
- `List (const List< T > &list)`  
*Copy constructor.*
- `List (List< T > &&list)`  
*Move constructor.*
- `List< T > & operator= (std::initializer_list< T > list)`  
*Copy assign the container with the contents of the initializer list.*
- `List< T > & operator= (const List< T > &list)`  
*Copy assignment operator.*
- `List< T > & operator= (List< T > &&list)`  
*Move assignment operator.*
- `void swap (List< T > &other)`  
*Swaps the contents of the list with those of other.*
- `bool empty () const`  
*Check whether the container is empty.*
- `std::size_t size () const`  
*Returns the size of the container.*
- `void clear ()`  
*Frees all elements in the container.*

## Protected Member Functions

- `iterator begin ()`
- `const_iterator begin () const`
- `iterator end ()`
- `const_iterator end () const`
- `T & operator[] (std::size_t index)`  
*Returns the reference to the element at the given index.*
- `const T & operator[] (std::size_t index) const`  
*Returns the reference to the element at the given index.*
- `T & at (std::size_t index)`  
*Returns the reference to the element at the given index.*
- `const T & at (std::size_t index) const`  
*Returns the reference to the element at the given index.*
- `void push_front (const T &value)`  
*Pushes the element to the front of the container.*
- `void push_back (const T &value)`  
*Pushes the element to the back of the container.*

- void `pop_front ()`  
*Removes the element at the front of the container.*
- void `pop_back ()`  
*Removes the element at the back of the container.*
- `iterator remove (const iterator &it)`  
*Removes the element iterator in the container.*
- int `remove (const T &value, const iterator &begin, const iterator &end)`  
*Removes the elements in the range [begin, end]*
- int `remove (const T &value)`  
*Removes the elements that has the same value as the given one.*
- int `remove_if (std::function< bool(const T &) > predicate, const iterator &begin, const iterator &end)`
- int `remove_if (std::function< bool(const T &) > predicate)`
- void `resize (std::size_t count)`  
*Resizes the container to contain the given number of elements.*
- void `resize (std::size_t count, const T &value)`  
*Resizes the container to contain the given number of elements.*
- void `assign (std::size_t count, const T &value)`  
*Assigns new contents to the list, replacing its current content with count copies of value*
- void `assign (const const_iterator &begin, const const_iterator &end)`  
*Assigns new contents to the list, replacing its current content.*
- void `assign (std::initializer_list< T > list)`  
*Assigns new contents to the list, replacing its current content.*
- std::size\_t `unique ()`  
*Eliminates all except the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.*
- std::size\_t `unique (std::function< bool(const T &, const T &) > predicate)`  
*Eliminates all except the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.*
- void `reverse ()`  
*Reverses the order of the elements in the list.*
- void `swap (List< T > &other)`  
*Swaps the contents of the list with those of other.*

## Private Types

- using `List = Core::List< T >`

## Private Member Functions

- void `reset ()`
- void `insert_previous (const iterator &it, const iterator &it_prev)`  
*Insert an iterator before the specified iterator.*
- `iterator move_previous (const iterator &it, const iterator &first, const iterator &last)`
- `iterator get_iterator (std::size_t index)`  
*Returns an iterator to the element at index index*

## Private Attributes

- `iterator mBegin`  
*The head of the list.*
- `iterator mEnd`  
*The end of the list (one past the last element)*
- `std::size_t mSize`  
*The size of the list.*

### 7.47.1 Detailed Description

```
template<typename T>
class Core::Queue< T >
```

The queue container.

#### Template Parameters

<code>T</code>	the type of the elements
----------------	--------------------------

### 7.47.2 Member Typedef Documentation

#### 7.47.2.1 List

```
template<typename T >
using Core::Queue< T >::List = Core::List< T > [private]
```

### 7.47.3 Member Function Documentation

#### 7.47.3.1 assign() [1/3]

```
template<typename T >
void Core::List< T >::assign (
    const const_iterator & begin,
    const const_iterator & end ) [inline], [protected], [inherited]
```

Assigns new contents to the list, replacing its current content.

#### Parameters

<code>begin</code>	The iterator to the first element to be assigned
<code>end</code>	The iterator to the last element to be assigned

### 7.47.3.2 assign() [2/3]

```
template<typename T >
void Core::List< T >::assign (
    std::initializer_list< T > list ) [inline], [protected], [inherited]
```

Assigns new contents to the list, replacing its current content.

#### Parameters

<i>list</i>	The initializer list to be assigned
-------------	-------------------------------------

### 7.47.3.3 assign() [3/3]

```
template<typename T >
void Core::List< T >::assign (
    std::size_t count,
    const T & value ) [inline], [protected], [inherited]
```

Assigns new contents to the list, replacing its current content with `count` copies of `value`.

#### Parameters

<i>count</i>	The new size of the container
<i>value</i>	The value to be used to fill the container

### 7.47.3.4 at() [1/2]

```
template<typename T >
T & Core::List< T >::at (
    std::size_t index ) [inline], [protected], [inherited]
```

Returns the reference to the element at the given index.

#### Parameters

<i>index</i>	The index of the element
--------------	--------------------------

#### Returns

T& The reference to the element at the given index

**Exceptions**

std::out\_of\_range: index out of range

**7.47.3.5 at() [2/2]**

```
template<typename T >
const T & Core::List< T >::at (
    std::size_t index ) const [inline], [protected], [inherited]
```

Returns the reference to the element at the given index.

**Parameters**

<i>index</i>	The index of the element
--------------	--------------------------

**Returns**

T& The reference to the element at the given index

**Exceptions**

std::out\_of\_range: index out of range

**7.47.3.6 back() [1/2]**

```
template<typename T >
T & Core::Queue< T >::back ( ) [inline]
```

Returns the reference to the back element of the queue.

**Returns**

T& the reference to the back element of the queue

**7.47.3.7 back() [2/2]**

```
template<typename T >
const T & Core::Queue< T >::back ( ) const [inline]
```

Returns the const reference to the back element of the queue.

**Returns**

const T& the const reference to the back element of the queue

### 7.47.3.8 begin() [1/2]

```
template<typename T >
iterator Core::List< T >::begin ( ) [inline], [protected], [inherited]
```

### 7.47.3.9 begin() [2/2]

```
template<typename T >
const_iterator Core::List< T >::begin ( ) const [inline], [protected], [inherited]
```

### 7.47.3.10 clear()

```
template<typename T >
void Core::List< T >::clear ( ) [inline], [inherited]
```

Frees all elements in the container.

### 7.47.3.11 empty()

```
template<typename T >
bool Core::List< T >::empty ( ) const [inline]
```

Check whether the container is empty.

#### Return values

<i>true</i>	The container is empty
<i>false</i>	The container is not empty

### 7.47.3.12 end() [1/2]

```
template<typename T >
iterator Core::List< T >::end ( ) [inline], [protected], [inherited]
```

### 7.47.3.13 end() [2/2]

```
template<typename T >
const_iterator Core::List< T >::end ( ) const [inline], [protected], [inherited]
```

#### 7.47.3.14 `front()` [1/2]

```
template<typename T >
T & Core::Queue< T >::front ( ) [inline]
```

Returns the reference to the front element of the queue.

##### Returns

T& the reference to the front element of the queue

#### 7.47.3.15 `front()` [2/2]

```
template<typename T >
const T & Core::Queue< T >::front ( ) const [inline]
```

Returns the const reference to the front element of the queue.

##### Returns

const T& the const reference to the front element of the queue

#### 7.47.3.16 `get_iterator()`

```
template<typename T >
iterator Core::List< T >::get_iterator (
    std::size_t index ) [inline], [private], [inherited]
```

Returns an iterator to the element at index `index`

##### Parameters

<code>index</code>	The index of the element
--------------------	--------------------------

##### Returns

An iterator to the element at index `index`

##### Exceptions

<code>std::out_of_range</code>	if `index` is out of range
--------------------------------	----------------------------

### 7.47.3.17 insert\_previous()

```
template<typename T >
void Core::List< T >::insert_previous (
    const iterator & it,
    const iterator & it_prev ) [inline], [private], [inherited]
```

Insert an iterator before the specified iterator.

#### Parameters

<i>it</i>	The iterator to insert the iterator (prev) before
<i>it_prev</i>	The iterator to insert

### 7.47.3.18 List() [1/4]

```
template<typename T >
Core::List< T >::List ( ) [inline]
```

Default constructor.

### 7.47.3.19 List() [2/4]

```
template<typename T >
Core::List< T >::List (
    const List< T > & list ) [inline]
```

Copy constructor.

#### Parameters

<i>rhs</i>	The other container
------------	---------------------

### 7.47.3.20 List() [3/4]

```
template<typename T >
Core::List< T >::List (
    List< T > && list ) [inline]
```

Move constructor.

**Parameters**

<i>rhs</i>	The other container
------------	---------------------

**7.47.3.21 List() [4/4]**

```
template<typename T >
Core::List< T >::List (
    std::initializer_list< T > list ) [inline]
```

Constructs the container with the contents of the initializer list.

**Parameters**

<i>init_list</i>	The initializer list
------------------	----------------------

**7.47.3.22 move\_previous()**

```
template<typename T >
iterator Core::List< T >::move_previous (
    const iterator & it,
    const iterator & first,
    const iterator & last ) [inline], [private], [inherited]
```

**7.47.3.23 operator=() [1/3]**

```
template<typename T >
List< T > & Core::List< T >::operator= (
    const List< T > & list ) [inline]
```

Copy assignment operator.

**Parameters**

<i>rhs</i>	The other container
------------	---------------------

**7.47.3.24 operator=() [2/3]**

```
template<typename T >
List< T > & Core::List< T >::operator= (
    List< T > && list ) [inline]
```

Move assignment operator.

#### Parameters

<i>rhs</i>	The other container
------------	---------------------

### 7.47.3.25 operator=( ) [3/3]

```
template<typename T >
List< T > & Core::List< T >::operator= (
    std::initializer_list< T > list ) [inline]
```

Copy assign the container with the contents of the initializer list.

#### Parameters

<i>init_list</i>	The initializer list
------------------	----------------------

### 7.47.3.26 operator[]( ) [1/2]

```
template<typename T >
T & Core::List< T >::operator[] ( 
    std::size_t index ) [inline], [protected], [inherited]
```

Returns the reference to the element at the given index.

#### Parameters

<i>index</i>	The index of the element
--------------	--------------------------

#### Returns

T& The reference to the element at the given index

#### Exceptions



std::out\_of\_range: index out of range

### 7.47.3.27 operator[]( ) [2/2]

```
template<typename T >
```

```
const T & Core::List< T >::operator[] (
    std::size_t index ) const [inline], [protected], [inherited]
```

Returns the reference to the element at the given index.

#### Parameters

<i>index</i>	The index of the element
--------------	--------------------------

#### Returns

T& The reference to the element at the given index

#### Exceptions



std::out\_of\_range: index out of range

### 7.47.3.28 **pop()**

```
template<typename T >
void Core::Queue< T >::pop () [inline]
```

Removes the front element of the queue.

### 7.47.3.29 **pop\_back()**

```
template<typename T >
void Core::List< T >::pop_back () [inline], [protected], [inherited]
```

Removes the element at the back of the container.

### 7.47.3.30 **pop\_front()**

```
template<typename T >
void Core::List< T >::pop_front () [inline], [protected], [inherited]
```

Removes the element at the front of the container.

### 7.47.3.31 **push() [1/2]**

```
template<typename T >
void Core::Queue< T >::push (
    const T & value ) [inline]
```

Inserts the element at the back of the queue.

**Parameters**

<code>elem</code>	The element to insert
-------------------	-----------------------

**7.47.3.32 push() [2/2]**

```
template<typename T >
void Core::Queue< T >::push (
    T && value ) [inline]
```

Inserts the element at the back of the queue.

**Parameters**

<code>elem</code>	The element to insert
-------------------	-----------------------

**7.47.3.33 push\_back()**

```
template<typename T >
void Core::List< T >::push_back (
    const T & value ) [inline], [protected], [inherited]
```

Pushes the element to the back of the container.

**Parameters**

<code>elem</code>	The element to be pushed into the back
-------------------	--

**7.47.3.34 push\_front()**

```
template<typename T >
void Core::List< T >::push_front (
    const T & value ) [inline], [protected], [inherited]
```

Pushes the element to the front of the container.

**Parameters**

<code>elem</code>	The element to be pushed into the front
-------------------	---

### 7.47.3.35 remove() [1/3]

```
template<typename T >
iterator Core::List< T >::remove (
    const iterator & it ) [inline], [protected], [inherited]
```

Removes the element iterator in the container.

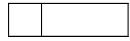
#### Parameters

<i>it</i>	The iterator to the element to be removed
-----------	---

#### Returns

iterator The iterator to the next element

#### Exceptions



std::out\_of\_range: iterator out of range

### 7.47.3.36 remove() [2/3]

```
template<typename T >
int Core::List< T >::remove (
    const T & value ) [inline], [protected], [inherited]
```

Removes the elements that has the same value as the given one.

#### Parameters

<i>value</i>	The value of the elements to be removed
--------------	---

#### Returns

iterator The iterator to the next element

### 7.47.3.37 remove() [3/3]

```
template<typename T >
int Core::List< T >::remove (
    const T & value,
    const iterator & begin,
    const iterator & end ) [inline], [protected], [inherited]
```

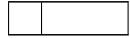
Removes the elements in the range [begin, end)

**Parameters**

<i>begin</i>	The iterator to the first element to be removed
<i>end</i>	The iterator to the last element to be removed

**Returns**

iterator The iterator to the next element

**Exceptions**

std::out\_of\_range: iterator out of range

**7.47.3.38 remove\_if() [1/2]**

```
template<typename T >
int Core::List< T >::remove_if (
    std::function< bool(const T &) > predicate ) [inline], [protected], [inherited]
```

**7.47.3.39 remove\_if() [2/2]**

```
template<typename T >
int Core::List< T >::remove_if (
    std::function< bool(const T &) > predicate,
    const iterator & begin,
    const iterator & end ) [inline], [protected], [inherited]
```

**7.47.3.40 reset()**

```
template<typename T >
void Core::List< T >::reset ( ) [inline], [private], [inherited]
```

**7.47.3.41 resize() [1/2]**

```
template<typename T >
void Core::List< T >::resize (
    std::size_t count ) [inline], [protected], [inherited]
```

Resizes the container to contain the given number of elements.

**Parameters**

<i>count</i>	The new size of the container
--------------	-------------------------------

**7.47.3.42 resize() [2/2]**

```
template<typename T >
void Core::List< T >::resize (
    std::size_t count,
    const T & value )  [inline], [protected], [inherited]
```

Resizes the container to contain the given number of elements.

**Parameters**

<i>count</i>	The new size of the container
<i>value</i>	The value to be used to fill the container

**7.47.3.43 reverse()**

```
template<typename T >
void Core::List< T >::reverse ()  [inline], [protected], [inherited]
```

Reverses the order of the elements in the list.

**7.47.3.44 size()**

```
template<typename T >
std::size_t Core::List< T >::size () const  [inline]
```

Returns the size of the container.

**Returns**

The size of the container

**7.47.3.45 swap() [1/2]**

```
template<typename T >
void Core::List< T >::swap (
    List< T > & other )  [inline], [protected], [inherited]
```

Swaps the contents of the list with those of other.

**Parameters**

<i>other</i>	list to exchange the contents with
--------------	------------------------------------

**7.47.3.46 swap() [2/2]**

```
template<typename T >
void Core::List< T >::swap (
    List< T > & other ) [inline]
```

Swaps the contents of the list with those of other.

**Parameters**

<i>other</i>	list to exchange the contents with
--------------	------------------------------------

**7.47.3.47 unique() [1/2]**

```
template<typename T >
std::size_t Core::List< T >::unique ( ) [inline], [protected], [inherited]
```

Eliminates all except the first element from every consecutive group of equivalent elements from the range [*first*, *last*) and returns a past-the-end iterator for the new logical end of the range.

**Returns**

the resulting size

**7.47.3.48 unique() [2/2]**

```
template<typename T >
std::size_t Core::List< T >::unique (
    std::function< bool(const T &, const T &) > predicate ) [inline], [protected],
[inherited]
```

Eliminates all except the first element from every consecutive group of equivalent elements from the range [*first*, *last*) and returns a past-the-end iterator for the new logical end of the range.

**Parameters**

<i>predicate</i>	Binary predicate that, taking two values of the same type than those contained in the list, returns true to remove the element passed as first argument from the container, and false otherwise.
------------------	--

**Returns**

the resulting size

## 7.47.4 Member Data Documentation

### 7.47.4.1 mBegin

```
template<typename T >
iterator Core::List< T >::mBegin  [private], [inherited]
```

The head of the list.

### 7.47.4.2 mEnd

```
template<typename T >
iterator Core::List< T >::mEnd  [private], [inherited]
```

The end of the list (one past the last element)

### 7.47.4.3 mSize

```
template<typename T >
std::size_t Core::List< T >::mSize  [private], [inherited]
```

The size of the list.

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

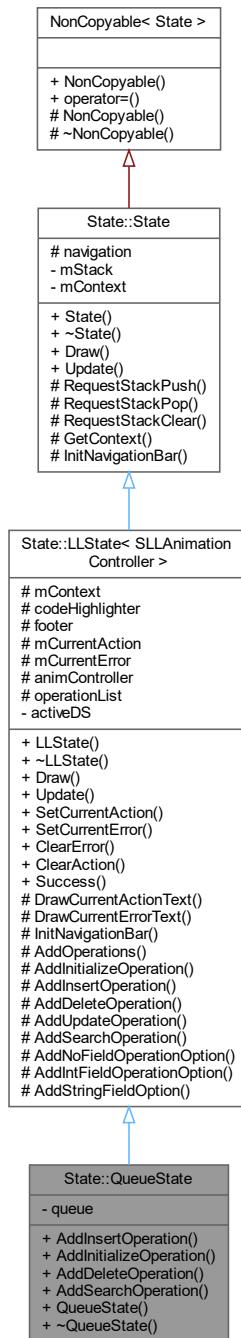
- src/Core/[Queue.hpp](#)

## 7.48 State::QueueState Class Reference

The state that is used to visualize the queue.

```
#include <QueueState.hpp>
```

Inheritance diagram for State::QueueState:



## Public Types

- `typedef std::unique_ptr< State > Ptr`

*A unique pointer to the state object.*

## Public Member Functions

- `void AddInsertOperation ()`  
*Add an insert operation to the queue algorithm.*
- `void AddInitializeOperation ()`  
*Add an initialize operation to the queue algorithm.*
- `void AddDeleteOperation ()`  
*Add an update operation to the queue algorithm.*
- `void AddSearchOperation ()`  
*Add a delete operation to the queue algorithm.*
- `QueueState (StateStack &stack, Context context)`  
*Construct a new QueueState object.*
- `~QueueState ()`  
*Destroy the QueueState object.*
- `virtual void Draw ()`  
*Draw the linked list state to the screen.*
- `virtual bool Update (float dt)`  
*Update the linked list state.*
- `virtual void SetCurrentAction (std::string action)`  
*Set the current action text.*
- `virtual void SetCurrentError (std::string error)`  
*Set the current error text.*
- `virtual void ClearError ()`  
*Clear the current error text.*
- `virtual void ClearAction ()`  
*Clear the current action text.*
- `virtual void Success ()`  
*Clear the current error and toggle the action list.*

## Protected Member Functions

- `virtual void DrawCurrentActionText ()`
- `virtual void DrawCurrentErrorText ()`
- `void InitNavigationBar ()`
- `virtual void AddOperations ()`  
*Add the operations to the operation list.*
- `virtual void AddUpdateOperation ()`  
*Add the update operation to the operation list.*
- `virtual void AddNoFieldOperationOption (GUIComponent::OperationContainer::Ptr container, std::string title, std::function< void() > action)`
- `virtual void AddIntFieldOperationOption (GUIComponent::OperationContainer::Ptr container, std::string title, Core::Deque< IntegerInput > fields, std::function< void(std::map< std::string, std::string >) > action)`
- `virtual void AddStringFieldOption (GUIComponent::OperationContainer::Ptr container, std::string title, std::string label, std::function< void(std::map< std::string, std::string >) > action)`
- `void RequestStackPush (States::ID stateID)`

- Request the state stack to push a new state/scene.  
• void [RequestStackPop \(\)](#)  
*Request the state stack to pop the current state/scene.*
- void [RequestStackClear \(\)](#)  
*Request the state stack to clear all the states/scenes.*
- [Context GetContext \(\) const](#)  
*Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).*

## Protected Attributes

- [Context mContext](#)
- [GUIComponent::CodeHighlighter::Ptr codeHighlighter](#)
- [GUIComponent::Footer< SLLAnimationController > footer](#)
- std::string [mCurrentAction](#)
- std::string [mCurrentError](#)
- T::Ptr [animController](#)
- [GUIComponent::OperationList operationList](#)
- [GUIComponent::NavigationBar navigation](#)

## Private Attributes

- [Algorithm::Queue queue](#)  
*The algorithm of the queue.*
- [DataStructures::ID activeDS](#)
- [StateStack \\* mStack](#)  
*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*

### 7.48.1 Detailed Description

The state that is used to visualize the queue.

#### Note

This Queue state is using the Singly Linked List [Animation](#) Controller as it can be visualized using the same animation controller.

### 7.48.2 Member Typedef Documentation

#### 7.48.2.1 Ptr

```
typedef std::unique_ptr< State > State::State::Ptr [inherited]
```

A unique pointer to the state object.

### 7.48.3 Constructor & Destructor Documentation

#### 7.48.3.1 QueueState()

```
State::QueueState::QueueState (
    StateStack & stack,
    Context context )
```

Construct a new [QueueState](#) object.

##### Parameters

<i>stack</i>	The state stack where the queue state is pushed to.
<i>context</i>	The context of the application.

#### 7.48.3.2 ~QueueState()

```
State::QueueState::~QueueState ( )
```

Destroy the [QueueState](#) object.

### 7.48.4 Member Function Documentation

#### 7.48.4.1 AddDeleteOperation()

```
void State::QueueState::AddDeleteOperation ( ) [virtual]
```

Add an update operation to the queue algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.48.4.2 AddInitializeOperation()

```
void State::QueueState::AddInitializeOperation ( ) [virtual]
```

Add an initialize operation to the queue algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.48.4.3 AddInsertOperation()

```
void State::QueueState::AddInsertOperation ( ) [virtual]
```

Add an insert operation to the queue algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.48.4.4 AddIntFieldOperationOption()

```
void State::LLState< SLLAnimationController >::AddIntFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    Core::Deque< IntegerInput > fields,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.48.4.5 AddNoFieldOperationOption()

```
void State::LLState< SLLAnimationController >::AddNoFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::function< void() > action ) [protected], [virtual], [inherited]
```

#### 7.48.4.6 AddOperations()

```
void State::LLState< SLLAnimationController >::AddOperations [protected], [virtual], [inherited]
```

Add the operations to the operation list.

##### Note

This function should not be overridden as it calls the other Add\*Operation functions.

#### 7.48.4.7 AddSearchOperation()

```
void State::QueueState::AddSearchOperation ( ) [virtual]
```

Add a delete operation to the queue algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.48.4.8 AddStringFieldOption()

```
void State::LLState< SLLAnimationController >::AddStringFieldOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::string label,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.48.4.9 AddUpdateOperation()

```
void State::LLState< SLLAnimationController >::AddUpdateOperation [protected], [virtual],
[inherited]
```

Add the update operation to the operation list.

Reimplemented in [State::SLLState](#).

#### 7.48.4.10 ClearAction()

```
void State::LLState< SLLAnimationController >::ClearAction [inline], [virtual], [inherited]
```

Clear the current action text.

#### 7.48.4.11 ClearError()

```
void State::LLState< SLLAnimationController >::ClearError [inline], [virtual], [inherited]
```

Clear the current error text.

#### 7.48.4.12 Draw()

```
void State::LLState< SLLAnimationController >::Draw [inline], [virtual], [inherited]
```

Draw the linked list state to the screen.

Implements [State::State](#).

#### 7.48.4.13 DrawCurrentActionText()

```
void State::LLState< SLLAnimationController >::DrawCurrentActionText [inline], [protected],  
[virtual], [inherited]
```

#### 7.48.4.14 DrawCurrentErrorText()

```
void State::LLState< SLLAnimationController >::DrawCurrentErrorText [inline], [protected],  
[virtual], [inherited]
```

#### 7.48.4.15 GetContext()

```
State::State::Context State::State::GetContext () const [protected], [inherited]
```

Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### Returns

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### See also

[Context](#)

#### 7.48.4.16 InitNavigationBar()

```
void State::LLState< SLLAnimationController >::InitNavigationBar [protected], [inherited]
```

#### 7.48.4.17 RequestStackClear()

```
void State::State::RequestStackClear () [protected], [inherited]
```

Request the state stack to clear all the states/scenes.

##### See also

[StateStack::ClearStates](#)

#### 7.48.4.18 RequestStackPop()

```
void State::State::RequestStackPop ( ) [protected], [inherited]
```

Request the state stack to pop the current state/scene.

See also

[StateStack::PopState](#)

#### 7.48.4.19 RequestStackPush()

```
void State::State::RequestStackPush (
    States::ID stateID ) [protected], [inherited]
```

Request the state stack to push a new state/scene.

Parameters

<i>stateID</i>	The ID of the state/scene that will be pushed
----------------	---

See also

[StateStack::PushState](#)

#### 7.48.4.20 SetCurrentAction()

```
void State::LLState< SLLAnimationController >::SetCurrentAction (
    std::string action ) [inline], [virtual], [inherited]
```

Set the current action text.

Parameters

<i>action</i>	The current action text.
---------------	--------------------------

#### 7.48.4.21 SetCurrentError()

```
void State::LLState< SLLAnimationController >::SetCurrentError (
    std::string error ) [inline], [virtual], [inherited]
```

Set the current error text.

**Parameters**

<code>error</code>	The current error text.
--------------------	-------------------------

**7.48.4.22 Success()**

```
void State::LLState< SLLAnimationController >::Success [inline], [virtual], [inherited]
```

Clear the current error and toggle the action list.

**7.48.4.23 Update()**

```
bool State::LLState< SLLAnimationController >::Update (
    float dt ) [virtual], [inherited]
```

Update the linked list state.

**Parameters**

<code>dt</code>	The delta time between two frames.
-----------------	------------------------------------

**Returns**

`true` If the update was successful.  
`false` If the update was unsuccessful.

Implements [State::State](#).

**7.48.5 Member Data Documentation****7.48.5.1 activeDS**

```
DataStructures::ID State::LLState< SLLAnimationController >::activeDS [private], [inherited]
```

**7.48.5.2 animController**

```
T::Ptr State::LLState< SLLAnimationController >::animController [protected], [inherited]
```

#### 7.48.5.3 codeHighlighter

```
GUIComponent::CodeHighlighter::Ptr State::LLState< SLLAnimationController >::codeHighlighter  
[protected], [inherited]
```

#### 7.48.5.4 footer

```
GUIComponent::Footer< SLLAnimationController > State::LLState< SLLAnimationController ><-  
::footer [protected], [inherited]
```

#### 7.48.5.5 mContext

```
Context State::LLState< SLLAnimationController >::mContext [protected], [inherited]
```

#### 7.48.5.6 mCurrentAction

```
std::string State::LLState< SLLAnimationController >::mCurrentAction [protected], [inherited]
```

#### 7.48.5.7 mCurrentError

```
std::string State::LLState< SLLAnimationController >::mCurrentError [protected], [inherited]
```

#### 7.48.5.8 mStack

```
StateStack* State::State::mStack [private], [inherited]
```

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

#### 7.48.5.9 navigation

```
GUIComponent::NavigationBar State::State::navigation [protected], [inherited]
```

### 7.48.5.10 operationList

```
GUIComponent::OperationList State::LLState< SLAnimationController >::operationList [protected],  
[inherited]
```

### 7.48.5.11 queue

```
Algorithm::Queue State::QueueState::queue [private]
```

The algorithm of the queue.

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

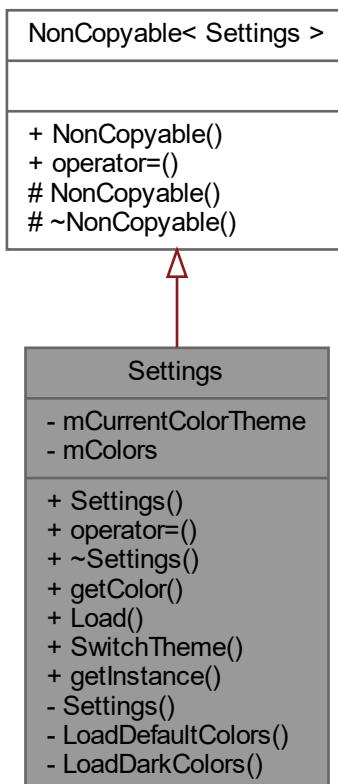
- src/States/LinkedList/QueueState.hpp
- src/States/LinkedList/QueueState.cpp

## 7.49 Settings Class Reference

The settings class that is used to store the settings.

```
#include <Settings.hpp>
```

Inheritance diagram for Settings:



## Public Member Functions

- `Settings (Settings &&)=delete`  
*The move constructor is deleted.*
- `Settings & operator= (Settings &&)=delete`  
*The assignment move operator is deleted.*
- `~Settings ()`  
*The destructor.*
- Color `getColor (std::size_t id) const`  
*The function that is used to get the color.*
- void `Load ()`  
*The function that is used to load the default color theme.*
- void `SwitchTheme ()`  
*The function that is used to switch the theme.*

## Static Public Member Functions

- static `Settings & getInstance ()`  
*The function that is used to get the instance of the settings class.*

## Private Member Functions

- `Settings ()=default`  
*The constructor.*
- void `LoadDefaultColors ()`
- void `LoadDarkColors ()`

## Private Attributes

- std::size\_t `mCurrentColorTheme`  
*The color struct that is used to store the color.*
- std::map< std::size\_t, Color > `mColors`  
*The colors that are used in the application.*

### 7.49.1 Detailed Description

The settings class that is used to store the settings.

The settings class is used to store the settings. Also, this class is using the singleton pattern to store the settings.

See also

[NonCopyable](#)  
[ColorTheme::ID](#)

### 7.49.2 Constructor & Destructor Documentation

### 7.49.2.1 Settings() [1/2]

```
Settings::Settings ( )  [private], [default]
```

The constructor.

The constructor that is used to initialize the settings class.

### 7.49.2.2 Settings() [2/2]

```
Settings::Settings (
    Settings && )  [delete]
```

The move constructor is deleted.

The move constructor is deleted to make the class non-copyable and satisfy the singleton pattern.

#### Parameters

<i>other</i>	The other settings class.
--------------	---------------------------

### 7.49.2.3 ~Settings()

```
Settings::~Settings ( )
```

The destructor.

The destructor that is used to destroy the settings class.

## 7.49.3 Member Function Documentation

### 7.49.3.1 getColor()

```
Color Settings::getColor (
    std::size_t id ) const
```

The function that is used to get the color.

The function that is used to get the color.

#### Parameters

<i>id</i>	The color ID.
-----------	---------------

**Returns**

Color The color.

**See also**

[ColorTheme::ID](#)

### 7.49.3.2 getInstance()

```
Settings & Settings::getInstance () [static]
```

The function that is used to get the instance of the settings class.

The function that is used to get the instance of the settings class.

**Returns**

Settings& A reference to the settings class.

### 7.49.3.3 Load()

```
void Settings::Load ()
```

The function that is used to load the default color theme.

The function that is used to load the default color theme.

### 7.49.3.4 LoadDarkColors()

```
void Settings::LoadDarkColors () [private]
```

### 7.49.3.5 LoadDefaultColors()

```
void Settings::LoadDefaultColors () [private]
```

### 7.49.3.6 operator=( )

```
Settings & Settings::operator= (
    Settings && ) [delete]
```

The assignment move operator is deleted.

The assignment move operator is deleted to make the class non-copyable and satisfy the singleton pattern.

**Parameters**

<i>other</i>	The other settings class.
--------------	---------------------------

**Returns**

[Settings](#)& A reference to the settings class.

### 7.49.3.7 SwitchTheme()

```
void Settings::SwitchTheme ( )
```

The function that is used to switch the theme.

The function that is used to switch the theme.

## 7.49.4 Member Data Documentation

### 7.49.4.1 mColors

```
std::map< std::size_t, Color > Settings::mColors [private]
```

The colors that are used in the application.

**See also**

[ColorTheme::ID](#)

### 7.49.4.2 mCurrentColorTheme

```
std::size_t Settings::mCurrentColorTheme [private]
```

The color struct that is used to store the color.

The color struct is used to store the color. This struct is used to store the color ID, the color name, and the color.

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

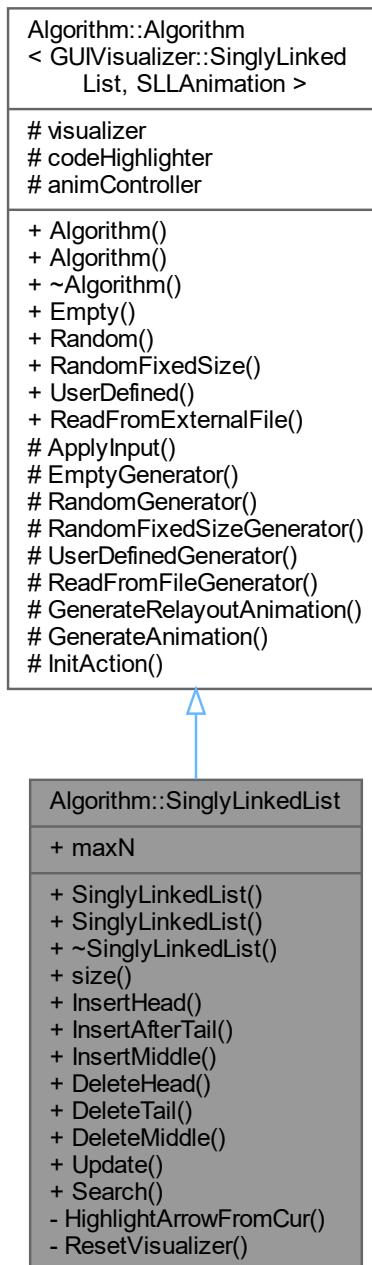
- src/[Settings.hpp](#)
- src/[Settings.cpp](#)

## 7.50 Algorithm::SinglyLinkedList Class Reference

The algorithm class that is used to generate step-by-step instructions for the visualization of the singly linked list.

```
#include <SinglyLinkedList.hpp>
```

Inheritance diagram for Algorithm::SinglyLinkedList:



## Public Member Functions

- `SinglyLinkedList ()`  
*The constructor of the singly linked list algorithm.*
- `SinglyLinkedList (GUIComponent::CodeHighlighter::Ptr codeHighlighter, SLLAnimationController::Ptr animController, FontHolder *fonts)`  
*The constructor of the singly linked list algorithm.*
- `~SinglyLinkedList ()`  
*The destructor of the singly linked list algorithm.*
- `std::size_t size () const`  
*Return the size of the singly linked list.*
- `void InsertHead (int value)`  
*Insert a new element before the head of the singly linked list.*
- `void InsertAfterTail (int value)`  
*Insert a new element after the tail of the singly linked list.*
- `void InsertMiddle (int index, int value)`  
*Insert a new element at the middle (not before the head or after the tail) of the singly linked list.*
- `void DeleteHead ()`  
*Delete the head of the singly linked list.*
- `void DeleteTail ()`  
*Delete the tail of the singly linked list.*
- `void DeleteMiddle (int index)`  
*Delete the middle (not the head or tail) of the singly linked list.*
- `void Update (int index, int value)`  
*Update the value of the head of the singly linked list.*
- `void Search (int value)`  
*Search for an element in the singly linked list.*
- `virtual void Empty ()`  
*Initialize an empty data structure, and generate animation.*
- `virtual void Random ()`  
*Initialize a data structure with random values input, and then generate animation.*
- `virtual void RandomFixedSize (int N)`  
*Initialize a fixed size data structure with random values input, and then generate animation.*
- `virtual void UserDefined (std::string input)`  
*Initialize a data structure with input from user, and then generate animation.*
- `virtual void ReadFromExternalFile (std::string path)`  
*Initialize a data structure with input from external file (which is used to store the input), and then generate animation.*

## Static Public Attributes

- `static constexpr int maxN = 16`  
*The maximum number of elements that the singly linked list can hold.*

## Protected Member Functions

- virtual void [ApplyInput](#) (std::vector< int > input, std::size\_t nMaxSize=10)
 

*Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an std::vector< int >) to the data structure, used by other initialization functions.*
- std::vector< int > [EmptyGenerator](#) ()
 

*Generate an empty input, this is used to generate the input for [Empty\(\)](#)*
- std::vector< int > [RandomGenerator](#) ()
 

*Generate a random input, this is used to generate the input for [Random\(\)](#)*
- std::vector< int > [RandomFixedSizeGenerator](#) (int nSize)
 

*Generate a random input with fixed size, this is used to generate the input for [RandomFixedSize\(\)](#)*
- std::vector< int > [UserDefinedGenerator](#) (std::string input)
 

*Generate an input from user, this is used to generate the input for [UserDefined\(\)](#)*
- std::vector< int > [ReadFromFileGenerator](#) (std::string inputFile)
 

*Parse the input from external file, this is used to generate the input for [ReadFromExternalFile\(\)](#)*
- virtual void [GenerateRelayoutAnimation](#) (Vector2 newPosition)
 

*Generate the relayout animation (which reposition the data structure to a new position on the screen)*
- virtual [SLLAnimation GenerateAnimation](#) (float duration, int highlightLine, std::string actionDescription)
 

*Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)*
- virtual void [InitAction](#) (std::vector< std::string > code)
 

*Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.*

## Protected Attributes

- [GUIVisualizer::SinglyLinkedList](#) visualizer
 

*Visualizer for the algorithm (which is used to draw animation generated by the algorithm)*
- [GUIComponent::CodeHighlighter::Ptr](#) codeHighlighter
 

*Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)*
- [Animation::AnimationController< SLLAnimation >::Ptr](#) animController
 

*Animation controller for the algorithm (which is used to control the animation generated by the algorithm)*

## Private Member Functions

- std::function< [GUIVisualizer::SinglyLinkedList](#)([GUIVisualizer::SinglyLinkedList](#), float, Vector2) > [HighlightArrowFromCur](#) (int index, bool drawVisualizer=true, bool reverse=false)
 

*Generate a highlight arrow from the node at the given index to the next node.*
- void [ResetVisualizer](#) ()
 

*Reset the visualizer of the singly linked list algorithm.*

### 7.50.1 Detailed Description

The algorithm class that is used to generate step-by-step instructions for the visualization of the singly linked list.

### 7.50.2 Constructor & Destructor Documentation

### 7.50.2.1 SinglyLinkedList() [1/2]

```
Algorithm::SinglyLinkedList::SinglyLinkedList ( )
```

The constructor of the singly linked list algorithm.

### 7.50.2.2 SinglyLinkedList() [2/2]

```
Algorithm::SinglyLinkedList::SinglyLinkedList (
    GUIComponent::CodeHighlighter::Ptr codeHighlighter,
    SLLAnimationController::Ptr animController,
    FontHolder * fonts )
```

The constructor of the singly linked list algorithm.

#### Parameters

<i>codeHighlighter</i>	The code highlighter of the singly linked list algorithm.
<i>animController</i>	The animation controller of the singly linked list algorithm.
<i>fonts</i>	The fonts of the singly linked list algorithm.

### 7.50.2.3 ~SinglyLinkedList()

```
Algorithm::SinglyLinkedList::~SinglyLinkedList ( )
```

The destructor of the singly linked list algorithm.

#### Note

This destructor is not virtual because this class is not designed to be inherited.

## 7.50.3 Member Function Documentation

### 7.50.3.1 ApplyInput()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::ApplyInput (
    std::vector< int > input,
    std::size_t nMaxSize = 10 ) [protected], [virtual], [inherited]
```

Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an `std::vector< int >`) to the data structure, used by other initialization functions.

**Parameters**

<i>input</i>	the input (in std::vector< int >) to be applied to the data structure
<i>nMaxSize</i>	the maximum size of the data structure, if the input size is larger than nMaxSize, the input will be truncated to nMaxSize

**7.50.3.2 DeleteHead()**

```
void Algorithm::SinglyLinkedList::DeleteHead ( )
```

Delete the head of the singly linked list.

**7.50.3.3 DeleteMiddle()**

```
void Algorithm::SinglyLinkedList::DeleteMiddle (
    int index )
```

Delete the middle (not the head or tail) of the singly linked list.

**Parameters**

<i>index</i>	The index of the element to be deleted.
--------------	---

**Note**

This function is written as the visualization of deleting an element at the middle of the singly linked list is different from deleting an element at the head or tail of the singly linked list.

**7.50.3.4 DeleteTail()**

```
void Algorithm::SinglyLinkedList::DeleteTail ( )
```

Delete the tail of the singly linked list.

**7.50.3.5 Empty()**

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLAnimation >::Empty [virtual],
[inherited]
```

Initialize an empty data structure, and generate animation.

### 7.50.3.6 EmptyGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::←
EmptyGenerator [protected], [inherited]
```

Generate an empty input, this is used to generate the input for [Empty \(\)](#)

#### Returns

`std::vector< int >` the empty input

### 7.50.3.7 GenerateAnimation()

```
SLLAnimation Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::Generate←
Animation (
    float duration,
    int highlightLine,
    std::string actionDescription ) [protected], [virtual], [inherited]
```

Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)

#### Parameters

<code>duration</code>	the duration of the animation
<code>highlightLine</code>	the line to be highlighted in the code highlighter (if the line is -1, then no line will be highlighted)
<code>actionDescription</code>	the description of the action (which is used to display in the animation)

#### Returns

`AnimationState` the empty animation state (which currently only contains the metadata of the animation)

### 7.50.3.8 GenerateRelayoutAnimation()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::GenerateRelayout←
Animation (
    Vector2 newPosition ) [inline], [protected], [virtual], [inherited]
```

Generate the relayout animation (which reposition the data structure to a new position on the screen)

#### Parameters

<code>newPosition</code>	the new position of the data structure
--------------------------	--

### 7.50.3.9 HighlightArrowFromCur()

```
std::function< GUIVisualizer::SinglyLinkedList (GUIVisualizer::SinglyLinkedList, float, Vector2)
> Algorithm::SinglyLinkedList::HighlightArrowFromCur (
    int index,
    bool drawVisualizer = true,
    bool reverse = false ) [private]
```

Generate a highlight arrow from the node at the given index to the next node.

#### Parameters

<i>index</i>	The index of the node to be highlighted.
<i>drawVisualizer</i>	Whether to draw the visualizer, this is set to true by default.
<i>reverse</i>	Whether to reverse the animation (play the animation in reverse), this is set to false by default.

#### Returns

The animation of the highlight arrow.

### 7.50.3.10 InitAction()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::InitAction (
    std::vector< std::string > code ) [protected], [virtual], [inherited]
```

Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.

#### Note

This function is used to clear the animation controller and the code highlighter, and then initialize a new instruction

### 7.50.3.11 InsertAfterTail()

```
void Algorithm::SinglyLinkedList::InsertAfterTail (
    int value )
```

Insert a new element after the tail of the singly linked list.

#### Parameters

<i>value</i>	The value of the new element.
--------------	-------------------------------

### 7.50.3.12 InsertHead()

```
void Algorithm::SinglyLinkedList::InsertHead (
    int value )
```

Insert a new element before the head of the singly linked list.

#### Parameters

<i>value</i>	The value of the new element.
--------------	-------------------------------

### 7.50.3.13 InsertMiddle()

```
void Algorithm::SinglyLinkedList::InsertMiddle (
    int index,
    int value )
```

Insert a new element at the middle (not before the head or after the tail) of the singly linked list.

#### Parameters

<i>index</i>	The index of the new element.
<i>value</i>	The value of the new element.

#### Note

This function is written as the visualization of inserting a new element at the middle of the singly linked list is different from inserting a new element at the head or tail of the singly linked list.

### 7.50.3.14 Random()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::Random [virtual],
[inherited]
```

Initialize a data structure with random values input, and then generate animation.

### 7.50.3.15 RandomFixedSize()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::RandomFixedSize (
    int N ) [virtual], [inherited]
```

Initialize a fixed size data structure with random values input, and then generate animation.

**Parameters**

<i>N</i>	the size of the data structure
----------	--------------------------------

**7.50.3.16 RandomFixedSizeGenerator()**

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::←
RandomFixedSizeGenerator (
    int nSize ) [protected], [inherited]
```

Generate a random input with fixed size, this is used to generate the input for [RandomFixedSize\(\)](#)

**Parameters**

<i>nSize</i>	the size of the input
--------------	-----------------------

**Returns**

```
std::vector< int > the random input with fixed size
```

**7.50.3.17 RandomGenerator()**

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::←
RandomGenerator [protected], [inherited]
```

Generate a random input, this is used to generate the input for [Random\(\)](#)

**Returns**

```
std::vector< int > the random input
```

**7.50.3.18 ReadFromExternalFile()**

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::ReadFromExternal←
File (
    std::string path ) [virtual], [inherited]
```

Initialize a data structure with input from external file (which is used to store the input), and then generate animation.

**Parameters**

<i>path</i>	the path to the external file
-------------	-------------------------------

### 7.50.3.19 ReadFromFileGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::←
ReadFromFileGenerator (
    std::string inputFile ) [protected], [inherited]
```

Parse the input from external file, this is used to generate the input for [ReadFromExternalFile \(\)](#)

#### Parameters

<i>inputFile</i>	the path to the external file
------------------	-------------------------------

#### Returns

```
std::vector< int > the input from external file
```

### 7.50.3.20 ResetVisualizer()

```
void Algorithm::SinglyLinkedList::ResetVisualizer ( ) [private]
```

Reset the visualizer of the singly linked list algorithm.

#### Note

This function is used to reset all of the effects that are only used to visualize the algorithm (i.e highlight the elements that are currently iterated).

### 7.50.3.21 Search()

```
void Algorithm::SinglyLinkedList::Search (
    int value )
```

Search for an element in the singly linked list.

#### Parameters

<i>value</i>	The value of the element to be searched.
--------------	--

### 7.50.3.22 size()

```
std::size_t Algorithm::SinglyLinkedList::size () const
```

Return the size of the singly linked list.

#### Returns

The size of the singly linked list.

### 7.50.3.23 Update()

```
void Algorithm::SinglyLinkedList::Update (
    int index,
    int value )
```

Update the value of the head of the singly linked list.

#### Parameters

<code>value</code>	The new value of the head of the singly linked list.
--------------------	--

### 7.50.3.24 UserDefined()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::UserDefined (
    std::string input ) [virtual], [inherited]
```

Initialize a data structure with input from user, and then generate animation.

#### Parameters

<code>input</code>	the input from user
--------------------	---------------------

### 7.50.3.25 UserDefinedGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::UserDefinedGenerator (
    std::string input ) [protected], [inherited]
```

Generate an input from user, this is used to generate the input for [UserDefined\(\)](#)

**Parameters**

<i>input</i>	the input from user
--------------	---------------------

**Returns**

```
std::vector< int > the input from user
```

## 7.50.4 Member Data Documentation

### 7.50.4.1 animController

```
Animation::AnimationController< SLLAnimation >::Ptr Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::animController [protected], [inherited]
```

Animation controller for the algorithm (which is used to control the animation generated by the algorithm)

**Note**

This [Algorithm](#) class is mainly use the animation controller to add animation for the algorithm

### 7.50.4.2 codeHighlighter

```
GUIComponent::CodeHighlighter::Ptr Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::codeHighlighter [protected], [inherited]
```

Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)

### 7.50.4.3 maxN

```
constexpr int Algorithm::SinglyLinkedList::maxN = 16 [static], [constexpr]
```

The maximum number of elements that the singly linked list can hold.

#### 7.50.4.4 visualizer

```
GUIVisualizer::SinglyLinkedList Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::visualizer [protected], [inherited]
```

Visualizer for the algorithm (which is used to draw animation generated by the algorithm)

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

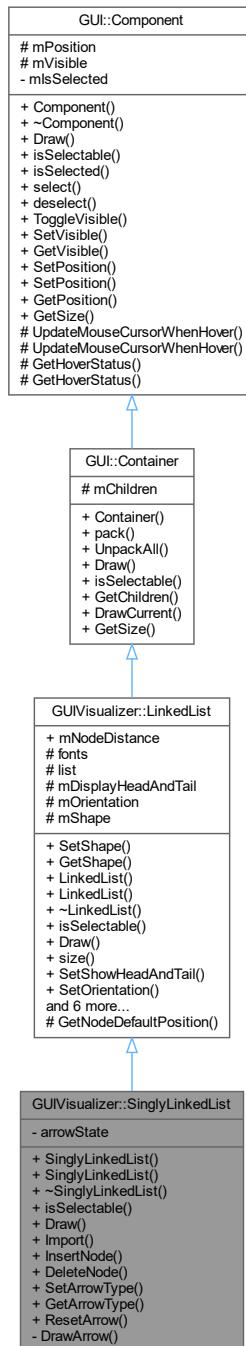
- src/Algorithms/LinkedList/[SinglyLinkedList.hpp](#)
- src/Algorithms/LinkedList/[SinglyLinkedList.cpp](#)

## 7.51 GUIVisualizer::SinglyLinkedList Class Reference

The singly linked list visualization.

```
#include <SinglyLinkedList.hpp>
```

Inheritance diagram for GUIVisualizer::SinglyLinkedList:



## Public Types

- enum `ArrowType` {
 `Default` , `Hidden` , `Active` , `Skip` ,
 `ArrowTypeCount` }
- The type of the arrow.*
- enum `Orientation` { `Horizontal` , `Vertical` , `OrientationCount` }

- The orientation of the linked list.*
- `typedef std::shared_ptr< Container > Ptr`  
*The shared pointer to the container.*
- ## Public Member Functions
- `SinglyLinkedList ()`  
*Construct a new Singly Linked List object.*
  - `SinglyLinkedList (FontHolder *fonts)`  
*Construct a new Singly Linked List object.*
  - `~SinglyLinkedList ()`  
*Destroy the Singly Linked List object.*
  - `bool isSelectable () const`  
*Check if the container is selectable.*
  - `void Draw (Vector2 base=(Vector2){0, 0}, float t=1.0f, bool init=false)`  
*Draw the singly linked list visualization.*
  - `void Import (std::vector< int > nodes)`  
*Initialize the singly linked list visualization with the given nodes.*
  - `void InsertNode (std::size_t index, Node node, bool rePosition=true)`  
*Insert a node into the singly linked list visualization.*
  - `void DeleteNode (std::size_t index, bool rePosition=true)`  
*Delete a node from the singly linked list visualization.*
  - `void SetArrowType (std::size_t index, ArrowType type)`
  - `ArrowType GetArrowType (std::size_t index)`
  - `void ResetArrow ()`
  - `void SetShape (Node::Shape shape)`  
*Set the shape of the node.*
  - `Node::Shape GetShape () const`  
*Get the shape of the node.*
  - `virtual void Draw (Vector2 base=(Vector2){0, 0})`  
*Draw the container.*
  - `virtual std::size_t size () const`  
*Get the size of the linked list visualization.*
  - `virtual void SetShowHeadAndTail (bool show)`
  - `virtual void SetOrientation (Orientation orientation)`  
*Set the orientation of the linked list visualization.*
  - `virtual std::vector< Node > & GetList ()`
  - `virtual Node GenerateNode (int value)`
  - `virtual void Relayout ()`  
*relayout the linked list visualization.*
  - `void pack (Component::Ptr component)`  
*Pack a component into the container.*
  - `void UnpackAll ()`  
*Unpack all components from the container.*
  - `Core::Deque< Component::Ptr > GetChildren ()`  
*Get the pack components of the container.*
  - `virtual void DrawCurrent (Vector2 base)`
  - `virtual Vector2 GetSize ()`  
*Get the size of the container.*
  - `bool isSelected () const`

- *Check if the component is selected.*
- virtual void [select \(\)](#)  
*Select the component.*
- virtual void [deselect \(\)](#)  
*Deselect the component.*
- virtual void [ToggleVisible \(\)](#)  
*Toggle the visibility of the component.*
- virtual void [SetVisible \(bool visible\)](#)  
*Set the visibility of the component.*
- virtual bool [GetVisible \(\)](#)  
*Get the visibility of the component.*
- void [SetPosition \(float x, float y\)](#)  
*Set the position of the component.*
- void [SetPosition \(Vector2 position\)](#)  
*Set the position of the component.*
- Vector2 [GetPosition \(\)](#)  
*Get the position of the component.*

## Static Public Attributes

- static constexpr float [mNodeDistance](#) = 20  
*The distance between the [GUI](#) nodes.*

## Protected Member Functions

- Vector2 [GetNodeDefaultPosition \(std::size\\_t index\)](#)
- virtual void [UpdateMouseCursorWhenHover \(std::map< std::string, Rectangle > bounds, bool hover, bool noHover\)](#)  
*Update the mouse cursor when hovering.*
- virtual void [UpdateMouseCursorWhenHover \(Rectangle bound, bool hover, bool noHover\)](#)  
*Update the mouse cursor when hovering.*
- virtual bool [GetHoverStatus \(std::map< std::string, Rectangle > bounds, bool hover, bool noHover\)](#)  
*Get the hover status of the component.*
- virtual bool [GetHoverStatus \(Rectangle bound, bool hover, bool noHover\)](#)  
*Get the hover status of the component.*

## Protected Attributes

- [FontHolder \\* fonts](#)
- std::vector< [Node](#) > [list](#)
- bool [mDisplayHeadAndTail](#)
- Orientation [mOrientation](#) = Orientation::Horizontal  
*The orientation of the linked list, by default it is horizontal.*
- [Node::Shape mShape](#) = [Node::Shape::Circle](#)  
*The shape of the node, by default it is a circle.*
- [Core::Deque< Component::Ptr > mChildren](#)
- Vector2 [mPosition](#)  
*The position of the component.*
- bool [mVisible](#)  
*The visibility of the component.*

## Private Member Functions

- void `DrawArrow` (Vector2 base, float t)

## Private Attributes

- std::vector< `ArrowType` > `arrowState`
- bool `mIsSelected`

*The selected status of the component.*

### 7.51.1 Detailed Description

The singly linked list visualization.

The singly linked list visualization is used to visualize the singly linked list.

### 7.51.2 Member Typedef Documentation

#### 7.51.2.1 Ptr

```
typedef std::shared_ptr< Container > GUI::Container::Ptr [inherited]
```

The shared pointer to the container.

The `GUI` container is commonly used by multiple components, so a shared pointer is used.

See also

`std::shared_ptr`

### 7.51.3 Member Enumeration Documentation

#### 7.51.3.1 ArrowType

```
enum GUIVisualizer::LinkedList::ArrowType [inherited]
```

The type of the arrow.

The type of the arrow is used to determine the color of the arrow.

**Enumerator**

Default	
Hidden	
Active	
Skip	
ArrowTypeCount	

**7.51.3.2 Orientation**

```
enum GUIVisualizer::LinkedList::Orientation [inherited]
```

The orientation of the linked list.

The orientation of the linked list is used to determine the orientation of the linked list.

**Enumerator**

Horizontal	
Vertical	
OrientationCount	

**7.51.4 Constructor & Destructor Documentation****7.51.4.1 SinglyLinkedList() [1/2]**

```
GUIVisualizer::SinglyLinkedList::SinglyLinkedList ( )
```

Construct a new Singly Linked List object.

**7.51.4.2 SinglyLinkedList() [2/2]**

```
GUIVisualizer::SinglyLinkedList::SinglyLinkedList (
    FontHolder * fonts )
```

Construct a new Singly Linked List object.

### 7.51.4.3 ~SinglyLinkedList()

```
GUIVisualizer::SinglyLinkedList::~SinglyLinkedList ( )
```

Destroy the Singly Linked List object.

## 7.51.5 Member Function Documentation

### 7.51.5.1 DeleteNode()

```
void GUIVisualizer::SinglyLinkedList::DeleteNode ( std::size_t index, bool rePosition = true ) [virtual]
```

Delete a node from the singly linked list visualization.

#### Parameters

<i>index</i>	The index to delete the node.
<i>rePosition</i>	Whether to reposition the whole singly linked list after deletion.

#### See also

[LinkedList::DeleteNode](#)

Reimplemented from [GUIVisualizer::LinkedList](#).

### 7.51.5.2 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

### 7.51.5.3 Draw() [1/2]

```
void GUI::Container::Draw ( Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Draw the container.

This function draws the container.

**Parameters**

<i>base</i>	The base position of the container.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#).

#### 7.51.5.4 Draw() [2/2]

```
void GUIVisualizer::SinglyLinkedList::Draw (
    Vector2 base = (Vector2){0, 0},
    float t = 1.0f,
    bool init = false )  [virtual]
```

Draw the singly linked list visualization.

This function draws the singly linked list visualization.

**Parameters**

<i>base</i>	The base position of the singly linked list visualization.
<i>t</i>	The time delta.
<i>init</i>	True if the singly linked list visualization is initializing.

Implements [GUIVisualizer::LinkedList](#).

#### 7.51.5.5 DrawArrow()

```
void GUIVisualizer::SinglyLinkedList::DrawArrow (
    Vector2 base,
    float t )  [private]
```

#### 7.51.5.6 DrawCurrent()

```
void GUI::Container::DrawCurrent (
    Vector2 base = (Vector2){0, 0} )  [virtual], [inherited]
```

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

### 7.51.5.7 GenerateNode()

```
GUIVisualizer::Node GUIVisualizer::LinkedList::GenerateNode (
    int value ) [virtual], [inherited]
```

### 7.51.5.8 GetArrowType()

```
GUIVisualizer::SinglyLinkedList::ArrowType GUIVisualizer::SinglyLinkedList::GetArrowType (
    std::size_t index )
```

### 7.51.5.9 GetChildren()

```
Core::Deque< GUI::Component::Ptr > GUI::Container::GetChildren () [inherited]
```

Get the pack components of the container.

This function returns the all of the pack components of the container.

#### Returns

The children of the container.

### 7.51.5.10 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

#### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

### 7.51.5.11 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
```

```
    bool hover,  
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

#### Parameters

<code>bounds</code>	The list of bound of the component.
---------------------	-------------------------------------

### 7.51.5.12 GetList()

```
std::vector< GUIVisualizer::Node > & GUIVisualizer::LinkedList::GetList ( ) [virtual], [inherited]
```

### 7.51.5.13 GetNodeDefaultPosition()

```
Vector2 GUIVisualizer::LinkedList::GetNodeDefaultPosition (   
    std::size_t index ) [protected], [inherited]
```

### 7.51.5.14GetPosition()

```
Vector2 GUI::Component::GetPosition ( ) [inherited]
```

Get the position of the component.

This function gets the position of the component.

#### Returns

The position of the component.

### 7.51.5.15 GetShape()

```
GUIVisualizer::Node::Shape GUIVisualizer::LinkedList::GetShape ( ) const [inherited]
```

Get the shape of the node.

### 7.51.5.16 GetSize()

```
Vector2 GUI::Container::GetSize ( ) [virtual], [inherited]
```

Get the size of the container.

This function returns the size of the container.

#### Returns

The size of the container.

Reimplemented from [GUI::Component](#).

Reimplemented in [GUIComponent::OperationList](#), and [GUIComponent::OptionInputField](#).

### 7.51.5.17 GetVisible()

```
bool GUI::Component::GetVisible ( ) [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

#### Returns

The visibility of the component.

### 7.51.5.18 Import()

```
void GUIVisualizer::SinglyLinkedList::Import ( std::vector< int > nodes ) [virtual]
```

Initialize the singly linked list visualization with the given nodes.

#### Parameters

<i>nodes</i>	The nodes to initialize the singly linked list visualization with.
--------------	--

#### See also

[LinkedList::Import](#)

Reimplemented from [GUIVisualizer::LinkedList](#).

### 7.51.5.19 InsertNode()

```
void GUIVisualizer::SinglyLinkedList::InsertNode (
    std::size_t index,
    GUIVisualizer::Node node,
    bool rePosition = true ) [virtual]
```

Insert a node into the singly linked list visualization.

#### Parameters

<i>index</i>	The index to insert the node.
<i>node</i>	The node to insert.
<i>rePosition</i>	Whether to reposition the whole singly linked list after insertion.

#### See also

[LinkedList::InsertNode](#)

Reimplemented from [GUIVisualizer::LinkedList](#).

### 7.51.5.20 isSelectable()

```
bool GUIVisualizer::SinglyLinkedList::isSelectable () const [virtual]
```

Check if the container is selectable.

**Deprecated** This function is deprecated.

This function checks if the container is selectable.

#### Returns

True if the container is selectable.

Reimplemented from [GUIVisualizer::LinkedList](#).

### 7.51.5.21 isSelected()

```
bool GUI::Component::isSelected () const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

#### Returns

True if the component is selected.

### 7.51.5.22 pack()

```
void GUI::Container::pack (
    Component::Ptr component ) [inherited]
```

Pack a component into the container.

This function packs a component into the container.

#### Parameters

<code>component</code>	The component to pack.
------------------------	------------------------

### 7.51.5.23 Relayout()

```
void GUIVisualizer::LinkedList::Relayout ( ) [virtual], [inherited]
```

relayout the linked list visualization.

### 7.51.5.24 ResetArrow()

```
void GUIVisualizer::SinglyLinkedList::ResetArrow ( )
```

### 7.51.5.25 select()

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

### 7.51.5.26 SetArrowType()

```
void GUIVisualizer::SinglyLinkedList::SetArrowType (
    std::size_t index,
    ArrowType type )
```

### 7.51.5.27 SetOrientation()

```
void GUIVisualizer::LinkedList::SetOrientation (
    Orientation orientation ) [virtual], [inherited]
```

Set the orientation of the linked list visualization.

**Parameters**

<i>orientation</i>	The orientation of the linked list visualization.
--------------------	---

**7.51.5.28 SetPosition() [1/2]**

```
void GUI::Component::SetPosition (
    float x,
    float y )  [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

**7.51.5.29 SetPosition() [2/2]**

```
void GUI::Component::SetPosition (
    Vector2 position )  [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>position</i>	The position of the component.
-----------------	--------------------------------

**7.51.5.30 SetShape()**

```
void GUIVisualizer::LinkedList::SetShape (
    Node::Shape shape )  [inherited]
```

Set the shape of the node.

### 7.51.5.31 SetShowHeadAndTail()

```
void GUIVisualizer::LinkedList::SetShowHeadAndTail (
    bool show ) [virtual], [inherited]
```

### 7.51.5.32 SetVisible()

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

#### Parameters

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

### 7.51.5.33 size()

```
std::size_t GUIVisualizer::LinkedList::size ( ) const [virtual], [inherited]
```

Get the size of the linked list visualization.

#### Returns

The size of the linked list visualization.

### 7.51.5.34 ToggleVisible()

```
void GUI::Component::ToggleVisible ( ) [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

### 7.51.5.35 UnpackAll()

```
void GUI::Container::UnpackAll ( ) [inherited]
```

Unpack all components from the container.

This function unpacks all components from the container.

#### See also

[pack](#)

### 7.51.5.36 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

#### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

### 7.51.5.37 UpdateMouseCursorWhenHover() [2/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

#### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

## 7.51.6 Member Data Documentation

### 7.51.6.1 arrowState

```
std::vector< ArrowType > GUIVisualizer::SinglyLinkedList::arrowState [private]
```

### 7.51.6.2 fonts

```
FontHolder* GUIVisualizer::LinkedList::fonts [protected], [inherited]
```

### 7.51.6.3 list

```
std::vector< Node > GUIVisualizer::LinkedList::list [protected], [inherited]
```

### 7.51.6.4 mChildren

```
Core::Deque< Component::Ptr > GUI::Container::mChildren [protected], [inherited]
```

### 7.51.6.5 mDisplayHeadAndTail

```
bool GUIVisualizer::LinkedList::mDisplayHeadAndTail [protected], [inherited]
```

### 7.51.6.6 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

### 7.51.6.7 mNodeDistance

```
constexpr float GUIVisualizer::LinkedList::mNodeDistance = 20 [static], [constexpr], [inherited]
```

The distance between the **GUI** nodes.

### 7.51.6.8 mOrientation

```
Orientation GUIVisualizer::LinkedList::mOrientation = Orientation::Horizontal [protected],  
[inherited]
```

The orientation of the linked list, by default it is horizontal.

### 7.51.6.9 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

### 7.51.6.10 mShape

```
Node::Shape GUIVisualizer::LinkedList::mShape = Node::Shape::Circle [protected], [inherited]
```

The shape of the node, by default it is a circle.

### 7.51.6.11 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

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

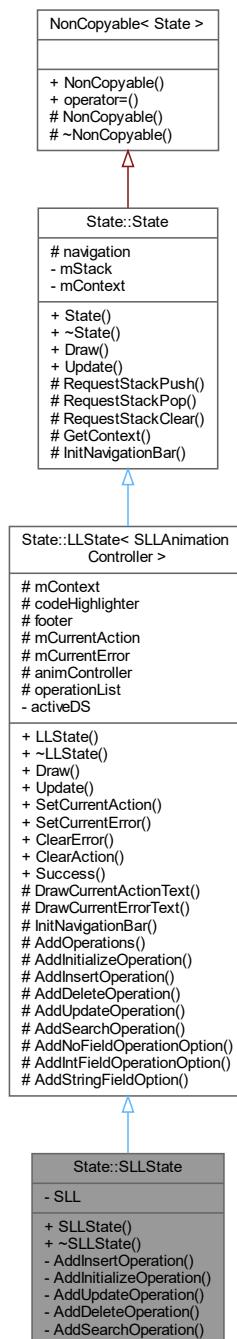
- src/Components/Visualization/[SinglyLinkedList.hpp](#)
- src/Components/Visualization/[SinglyLinkedList.cpp](#)

## 7.52 State::SLLState Class Reference

The state class that is used to represent the singly linked list state/scene of the application.

```
#include <SLLState.hpp>
```

Inheritance diagram for State::SLLState:



## Public Types

- `typedef std::unique_ptr< State > Ptr`  
*A unique pointer to the state object.*

## Public Member Functions

- `SLLState (StateStack &stack, Context context)`  
*Construct a new SLLState object.*
- `~SLLState ()`  
*Destroy the SLLState object.*
- `virtual void Draw ()`  
*Draw the linked list state to the screen.*
- `virtual bool Update (float dt)`  
*Update the linked list state.*
- `virtual void SetCurrentAction (std::string action)`  
*Set the current action text.*
- `virtual void SetCurrentError (std::string error)`  
*Set the current error text.*
- `virtual void ClearError ()`  
*Clear the current error text.*
- `virtual void ClearAction ()`  
*Clear the current action text.*
- `virtual void Success ()`  
*Clear the current error and toggle the action list.*

## Protected Member Functions

- `virtual void DrawCurrentActionText ()`
- `virtual void DrawCurrentErrorText ()`
- `void InitNavigationBar ()`
- `virtual void AddOperations ()`  
*Add the operations to the operation list.*
- `virtual void AddNoFieldOperationOption (GUIComponent::OperationContainer::Ptr container, std::string title, std::function< void() > action)`
- `virtual void AddIntFieldOperationOption (GUIComponent::OperationContainer::Ptr container, std::string title, Core::Deque< IntegerInput > fields, std::function< void(std::map< std::string, std::string >) > action)`
- `virtual void AddStringFieldOption (GUIComponent::OperationContainer::Ptr container, std::string title, std::string label, std::function< void(std::map< std::string, std::string >) > action)`
- `void RequestStackPush (States::ID stateID)`  
*Request the state stack to push a new state/scene.*
- `void RequestStackPop ()`  
*Request the state stack to pop the current state/scene.*
- `void RequestStackClear ()`  
*Request the state stack to clear all the states/scenes.*
- `Context GetContext () const`  
*Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).*

## Protected Attributes

- Context mContext
- GUIComponent::CodeHighlighter::Ptr codeHighlighter
- GUIComponent::Footer< SLLAnimationController > footer
- std::string mCurrentAction
- std::string mCurrentError
- T::Ptr animController
- GUIComponent::OperationList operationList
- GUIComponent::NavigationBar navigation

## Private Member Functions

- void [AddInsertOperation \(\)](#)  
*Add an insert operation to the singly linked list algorithm.*
- void [AddInitializeOperation \(\)](#)  
*Add an initialize operation to the singly linked list algorithm.*
- void [AddUpdateOperation \(\)](#)  
*Add an update operation to the singly linked list algorithm.*
- void [AddDeleteOperation \(\)](#)  
*Add a delete operation to the singly linked list algorithm.*
- void [AddSearchOperation \(\)](#)  
*Add a search operation to the singly linked list algorithm.*

## Private Attributes

- Algorithm::SinglyLinkedList SLL  
*The algorithm of the singly linked list.*
- DataStructures::ID activeDS
- StateStack \* mStack  
*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*

### 7.52.1 Detailed Description

The state class that is used to represent the singly linked list state/scene of the application.

### 7.52.2 Member Typedef Documentation

#### 7.52.2.1 Ptr

```
typedef std::unique_ptr< State > State::State::Ptr [inherited]
```

A unique pointer to the state object.

### 7.52.3 Constructor & Destructor Documentation

#### 7.52.3.1 SLLState()

```
State::SLLState::SLLState (
    StateStack & stack,
    Context context )
```

Construct a new [SLLState](#) object.

##### Parameters

<i>stack</i>	The state stack where the singly linked list state is pushed to.
<i>context</i>	The context of the application.

#### 7.52.3.2 ~SLLState()

```
State::SLLState::~SLLState ()
```

Destroy the [SLLState](#) object.

### 7.52.4 Member Function Documentation

#### 7.52.4.1 AddDeleteOperation()

```
void State::SLLState::AddDeleteOperation () [private], [virtual]
```

Add a delete operation to the singly linked list algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.52.4.2 AddInitializeOperation()

```
void State::SLLState::AddInitializeOperation () [private], [virtual]
```

Add an initialize operation to the singly linked list algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.52.4.3 AddInsertOperation()

```
void State::SLLState::AddInsertOperation ( ) [private], [virtual]
```

Add an insert operation to the singly linked list algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.52.4.4 AddIntFieldOperationOption()

```
void State::LLState< SLLAnimationController >::AddIntFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    Core::Deque< IntegerInput > fields,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.52.4.5 AddNoFieldOperationOption()

```
void State::LLState< SLLAnimationController >::AddNoFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::function< void() > action ) [protected], [virtual], [inherited]
```

#### 7.52.4.6 AddOperations()

```
void State::LLState< SLLAnimationController >::AddOperations [protected], [virtual], [inherited]
```

Add the operations to the operation list.

##### Note

This function should not be overridden as it calls the other Add\*Operation functions.

#### 7.52.4.7 AddSearchOperation()

```
void State::SLLState::AddSearchOperation ( ) [private], [virtual]
```

Add a search operation to the singly linked list algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.52.4.8 AddStringFieldOption()

```
void State::LLState< SLLAnimationController >::AddStringFieldOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::string label,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.52.4.9 AddUpdateOperation()

```
void State::SLLState::AddUpdateOperation () [private], [virtual]
```

Add an update operation to the singly linked list algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.52.4.10 ClearAction()

```
void State::LLState< SLLAnimationController >::ClearAction [inline], [virtual], [inherited]
```

Clear the current action text.

#### 7.52.4.11 ClearError()

```
void State::LLState< SLLAnimationController >::ClearError [inline], [virtual], [inherited]
```

Clear the current error text.

#### 7.52.4.12 Draw()

```
void State::LLState< SLLAnimationController >::Draw [inline], [virtual], [inherited]
```

Draw the linked list state to the screen.

Implements [State::State](#).

#### 7.52.4.13 DrawCurrentActionText()

```
void State::LLState< SLLAnimationController >::DrawCurrentActionText [inline], [protected],  
[virtual], [inherited]
```

#### 7.52.4.14 DrawCurrentErrorText()

```
void State::LLState< SLLAnimationController >::DrawCurrentErrorText [inline], [protected],  
[virtual], [inherited]
```

#### 7.52.4.15 GetContext()

```
State::State::Context State::State::GetContext () const [protected], [inherited]
```

Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### Returns

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### See also

[Context](#)

#### 7.52.4.16 InitNavigationBar()

```
void State::LLState< SLLAnimationController >::InitNavigationBar [protected], [inherited]
```

#### 7.52.4.17 RequestStackClear()

```
void State::State::RequestStackClear () [protected], [inherited]
```

Request the state stack to clear all the states/scenes.

##### See also

[StateStack::ClearStates](#)

#### 7.52.4.18 RequestStackPop()

```
void State::State::RequestStackPop ( ) [protected], [inherited]
```

Request the state stack to pop the current state/scene.

See also

[StateStack::PopState](#)

#### 7.52.4.19 RequestStackPush()

```
void State::State::RequestStackPush (
    States::ID stateID ) [protected], [inherited]
```

Request the state stack to push a new state/scene.

Parameters

<i>stateID</i>	The ID of the state/scene that will be pushed
----------------	---

See also

[StateStack::PushState](#)

#### 7.52.4.20 SetCurrentAction()

```
void State::LLState< SLLAnimationController >::SetCurrentAction (
    std::string action ) [inline], [virtual], [inherited]
```

Set the current action text.

Parameters

<i>action</i>	The current action text.
---------------	--------------------------

#### 7.52.4.21 SetCurrentError()

```
void State::LLState< SLLAnimationController >::SetCurrentError (
    std::string error ) [inline], [virtual], [inherited]
```

Set the current error text.

**Parameters**

<code>error</code>	The current error text.
--------------------	-------------------------

**7.52.4.22 Success()**

```
void State::LLState< SLLAnimationController >::Success [inline], [virtual], [inherited]
```

Clear the current error and toggle the action list.

**7.52.4.23 Update()**

```
bool State::LLState< SLLAnimationController >::Update (
    float dt ) [virtual], [inherited]
```

Update the linked list state.

**Parameters**

<code>dt</code>	The delta time between two frames.
-----------------	------------------------------------

**Returns**

`true` If the update was successful.  
`false` If the update was unsuccessful.

Implements [State::State](#).

**7.52.5 Member Data Documentation****7.52.5.1 activeDS**

```
DataStructures::ID State::LLState< SLLAnimationController >::activeDS [private], [inherited]
```

**7.52.5.2 animController**

```
T::Ptr State::LLState< SLLAnimationController >::animController [protected], [inherited]
```

### 7.52.5.3 codeHighlighter

```
GUIComponent::CodeHighlighter::Ptr State::LLState< SLLAnimationController >::codeHighlighter  
[protected], [inherited]
```

### 7.52.5.4 footer

```
GUIComponent::Footer< SLLAnimationController > State::LLState< SLLAnimationController ><-  
::footer [protected], [inherited]
```

### 7.52.5.5 mContext

```
Context State::LLState< SLLAnimationController >::mContext [protected], [inherited]
```

### 7.52.5.6 mCurrentAction

```
std::string State::LLState< SLLAnimationController >::mCurrentAction [protected], [inherited]
```

### 7.52.5.7 mCurrentError

```
std::string State::LLState< SLLAnimationController >::mCurrentError [protected], [inherited]
```

### 7.52.5.8 mStack

```
StateStack* State::State::mStack [private], [inherited]
```

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

### 7.52.5.9 navigation

```
GUIComponent::NavigationBar State::State::navigation [protected], [inherited]
```

### 7.52.5.10 operationList

```
GUIComponent::OperationList State::LLState< SLLAnimationController >::operationList [protected],  
[inherited]
```

### 7.52.5.11 SLL

```
Algorithm::SinglyLinkedList State::SLLState::SLL [private]
```

The algorithm of the singly linked list.

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

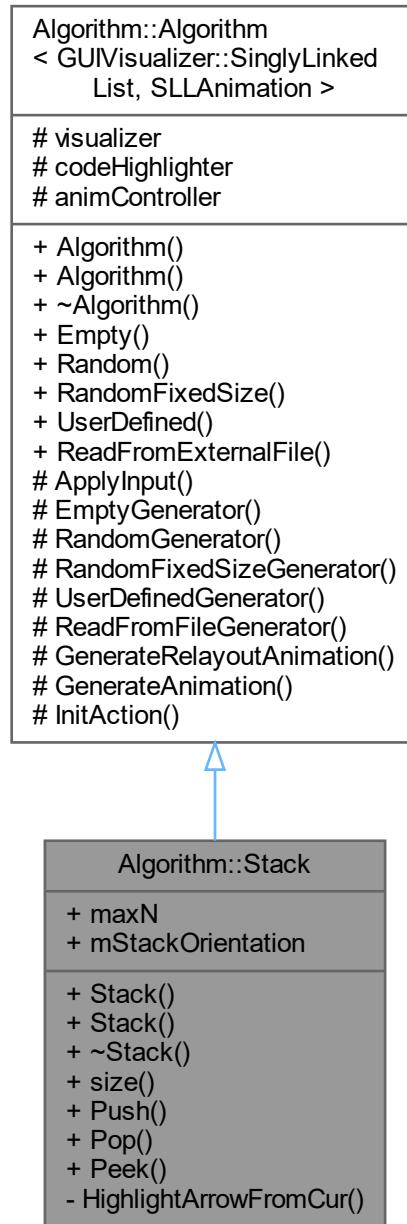
- src/States/LinkedList/[SLLState.hpp](#)
- src/States/LinkedList/[SLLState.cpp](#)

## 7.53 Algorithm::Stack Class Reference

The algorithm class that is used to generate step-by-step instructions for the visualization of the stack.

```
#include <Stack.hpp>
```

Inheritance diagram for Algorithm::Stack:



## Public Member Functions

- `Stack (GUICodeHighlighter::CodeHighlighter::Ptr codeHighlighter, SLLAnimationController::Ptr animController, FontHolder *fonts)`  
*The constructor of the stack algorithm.*
- `Stack ()`  
*The constructor of the stack algorithm.*

- `~Stack ()`  
*The destructor of the stack algorithm.*
- `std::size_t size () const`  
*Return the size of the stack.*
- `void Push (int value)`  
*Insert a new element before the head of the stack.*
- `void Pop ()`  
*Remove the element at the head of the stack.*
- `void Peek ()`  
*Return the value of the element at the head of the stack.*
- `virtual void Empty ()`  
*Initialize an empty data structure, and generate animation.*
- `virtual void Random ()`  
*Initialize a data structure with random values input, and then generate animation.*
- `virtual void RandomFixedSize (int N)`  
*Initialize a fixed size data structure with random values input, and then generate animation.*
- `virtual void UserDefined (std::string input)`  
*Initialize a data structure with input from user, and then generate animation.*
- `virtual void ReadFromExternalFile (std::string path)`  
*Initialize a data structure with input from external file (which is used to store the input), and then generate animation.*

## Static Public Attributes

- `static constexpr int maxN = 10`  
*The maximum number of elements that the stack can hold.*
- `static constexpr Orientation mStackOrientation = Orientation::Vertical`  
*The orientation of the stack, which is vertical.*

## Protected Member Functions

- `virtual void ApplyInput (std::vector< int > input, std::size_t nMaxSize=10)`  
*Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an std::vector< int >) to the data structure, used by other initialization functions.*
- `std::vector< int > EmptyGenerator ()`  
*Generate an empty input, this is used to generate the input for `Empty ()`*
- `std::vector< int > RandomGenerator ()`  
*Generate a random input, this is used to generate the input for `Random ()`*
- `std::vector< int > RandomFixedSizeGenerator (int nSize)`  
*Generate a random input with fixed size, this is used to generate the input for `RandomFixedSize ()`*
- `std::vector< int > UserDefinedGenerator (std::string input)`  
*Generate an input from user, this is used to generate the input for `UserDefined ()`*
- `std::vector< int > ReadFromFileGenerator (std::string inputFile)`  
*Parse the input from external file, this is used to generate the input for `ReadFromExternalFile ()`*
- `virtual void GenerateRelayoutAnimation (Vector2 newPosition)`  
*Generate the relayout animation (which reposition the data structure to a new position on the screen)*
- `virtual SLLAnimation GenerateAnimation (float duration, int highlightLine, std::string actionDescription)`  
*Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)*
- `virtual void InitAction (std::vector< std::string > code)`  
*Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.*

## Protected Attributes

- `GUIVisualizer::SinglyLinkedList visualizer`  
*Visualizer for the algorithm (which is used to draw animation generated by the algorithm)*
- `GUICOMPONENT::CodeHighlighter::Ptr codeHighlighter`  
*Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)*
- `Animation::AnimationController< SLLAnimation >::Ptr animController`  
*Animation controller for the algorithm (which is used to control the animation generated by the algorithm)*

## Private Member Functions

- `std::function< GUIVisualizer::SinglyLinkedList(GUIVisualizer::SinglyLinkedList, float, Vector2) > HighlightArrowFromCur`  
`(int index, bool drawVisualizer=true, bool reverse=false)`  
*Generate a highlight arrow from the node at the given index to the next node.*

### 7.53.1 Detailed Description

The algorithm class that is used to generate step-by-step instructions for the visualization of the stack.

### 7.53.2 Constructor & Destructor Documentation

#### 7.53.2.1 Stack() [1/2]

```
Algorithm::Stack::Stack (
    GUICOMPONENT::CodeHighlighter::Ptr codeHighlighter,
    SLLAnimationController::Ptr animController,
    FontHolder * fonts )
```

The constructor of the stack algorithm.

#### Parameters

<code>codeHighlighter</code>	The code highlighter of the stack algorithm.
<code>animController</code>	The animation controller of the stack algorithm.
<code>fonts</code>	The fonts of the stack algorithm.

#### 7.53.2.2 Stack() [2/2]

```
Algorithm::Stack::Stack ( )
```

The constructor of the stack algorithm.

### 7.53.2.3 ~Stack()

```
Algorithm::Stack::~Stack ( )
```

The destructor of the stack algorithm.

#### Note

This destructor is not virtual because this class is not designed to be inherited.

## 7.53.3 Member Function Documentation

### 7.53.3.1 ApplyInput()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::ApplyInput (
    std::vector< int > input,
    std::size_t nMaxSize = 10 ) [protected], [virtual], [inherited]
```

Apply an input to the data structure, it will then generate the animation function is used to apply an input (which is an `std::vector< int >`) to the data structure, used by other initialization functions.

#### Parameters

<code>input</code>	the input (in <code>std::vector&lt; int &gt;</code> ) to be applied to the data structure
<code>nMaxSize</code>	the maximum size of the data structure, if the input size is larger than <code>nMaxSize</code> , the input will be truncated to <code>nMaxSize</code>

### 7.53.3.2 Empty()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::Empty [virtual],
[inherited]
```

Initialize an empty data structure, and generate animation.

### 7.53.3.3 EmptyGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::EmptyGenerator [protected], [inherited]
```

Generate an empty input, this is used to generate the input for [Empty \(\)](#)

#### Returns

`std::vector< int >` the empty input

### 7.53.3.4 GenerateAnimation()

```
SLLAnimation Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::GenerateAnimation (
    float duration,
    int highlightLine,
    std::string actionDescription ) [protected], [virtual], [inherited]
```

Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)

#### Parameters

<i>duration</i>	the duration of the animation
<i>highlightLine</i>	the line to be highlighted in the code highlighter (if the line is -1, then no line will be highlighted)
<i>actionDescription</i>	the description of the action (which is used to display in the animation)

#### Returns

AnimationState the empty animation state (which currently only contains the metadata of the animation)

### 7.53.3.5 GenerateRelayoutAnimation()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::GenerateRelayoutAnimation (
    Vector2 newPosition ) [inline], [protected], [virtual], [inherited]
```

Generate the relayout animation (which reposition the data structure to a new position on the screen)

#### Parameters

<i>newPosition</i>	the new position of the data structure
--------------------	--

### 7.53.3.6 HighlightArrowFromCur()

```
std::function< GUIVisualizer::SinglyLinkedList(GUIVisualizer::SinglyLinkedList, float, Vector2) > Algorithm::Stack::HighlightArrowFromCur (
    int index,
    bool drawVisualizer = true,
    bool reverse = false ) [private]
```

Generate a highlight arrow from the node at the given index to the next node.

**Parameters**

<i>index</i>	The index of the node to be highlighted.
<i>drawVisualizer</i>	Whether to draw the visualizer, this is set to true by default.
<i>reverse</i>	Whether to reverse the animation (play the animation in reverse), this is set to false by default.

**Returns**

The animation of the highlight arrow.

**7.53.3.7 InitAction()**

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLAnimation >::InitAction (
    std::vector< std::string > code ) [protected], [virtual], [inherited]
```

Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.

**Note**

This function is used to clear the animation controller and the code highlighter, and then initialize a new instruction

**7.53.3.8 Peek()**

```
void Algorithm::Stack::Peek ( )
```

Return the value of the element at the head of the stack.

**Returns**

The value of the element at the head of the stack.

**7.53.3.9 Pop()**

```
void Algorithm::Stack::Pop ( )
```

Remove the element at the head of the stack.

**7.53.3.10 Push()**

```
void Algorithm::Stack::Push (
    int value )
```

Insert a new element before the head of the stack.

**Parameters**

<code>value</code>	The value of the new element.
--------------------	-------------------------------

**7.53.3.11 Random()**

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::Random [virtual],  
[inherited]
```

Initialize a data structure with random values input, and then generate animation.

**7.53.3.12 RandomFixedSize()**

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::RandomFixedSize (  
    int N ) [virtual], [inherited]
```

Initialize a fixed size data structure with random values input, and then generate animation.

**Parameters**

<code>N</code>	the size of the data structure
----------------	--------------------------------

**7.53.3.13 RandomFixedSizeGenerator()**

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::RandomFixedSizeGenerator (   
    int nSize ) [protected], [inherited]
```

Generate a random input with fixed size, this is used to generate the input for `RandomFixedSize()`

**Parameters**

<code>nSize</code>	the size of the input
--------------------	-----------------------

**Returns**

```
std::vector< int > the random input with fixed size
```

### 7.53.3.14 RandomGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::←
RandomGenerator [protected], [inherited]
```

Generate a random input, this is used to generate the input for [Random\(\)](#)

#### Returns

`std::vector< int >` the random input

### 7.53.3.15 ReadFromExternalFile()

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::ReadFromExternal←
File (
    std::string path ) [virtual], [inherited]
```

Initialize a data structure with input from external file (which is used to store the input), and then generate animation.

#### Parameters

<code>path</code>	the path to the external file
-------------------	-------------------------------

### 7.53.3.16 ReadFromFileGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::←
ReadFromFileGenerator (
    std::string inputFile ) [protected], [inherited]
```

Parse the input from external file, this is used to generate the input for [ReadFromExternalFile\(\)](#)

#### Parameters

<code>inputFile</code>	the path to the external file
------------------------	-------------------------------

#### Returns

`std::vector< int >` the input from external file

### 7.53.3.17 size()

```
std::size_t Algorithm::Stack::size ( ) const
```

Return the size of the stack.

**Returns**

The size of the stack.

**7.53.3.18 UserDefined()**

```
void Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::UserDefined ( std::string input ) [virtual], [inherited]
```

Initialize a data structure with input from user, and then generate animation.

**Parameters**

<i>input</i>	the input from user
--------------	---------------------

**7.53.3.19 UserDefinedGenerator()**

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::UserDefinedGenerator ( std::string input ) [protected], [inherited]
```

Generate an input from user, this is used to generate the input for [UserDefined\(\)](#)

**Parameters**

<i>input</i>	the input from user
--------------	---------------------

**Returns**

`std::vector< int >` the input from user

**7.53.4 Member Data Documentation****7.53.4.1 animController**

```
Animation::AnimationController< SLLAnimation >::Ptr Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation >::animController [protected], [inherited]
```

[Animation](#) controller for the algorithm (which is used to control the animation generated by the algorithm)

**Note**

This [Algorithm](#) class is mainly use the animation controller to add animation for the algorithm

#### 7.53.4.2 codeHighlighter

```
GUIComponent::CodeHighlighter::Ptr Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList ,  
SLLAnimation >::codeHighlighter [protected], [inherited]
```

Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)

#### 7.53.4.3 maxN

```
constexpr int Algorithm::Stack::maxN = 10 [static], [constexpr]
```

The maximum number of elements that the stack can hold.

#### 7.53.4.4 mStackOrientation

```
constexpr Orientation Algorithm::Stack::mStackOrientation = Orientation::Vertical [static],  
[constexpr]
```

The orientation of the stack, which is vertical.

#### 7.53.4.5 visualizer

```
GUIVisualizer::SinglyLinkedList Algorithm::Algorithm< GUIVisualizer::SinglyLinkedList , SLLAnimation  
>::visualizer [protected], [inherited]
```

Visualizer for the algorithm (which is used to draw animation generated by the algorithm)

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

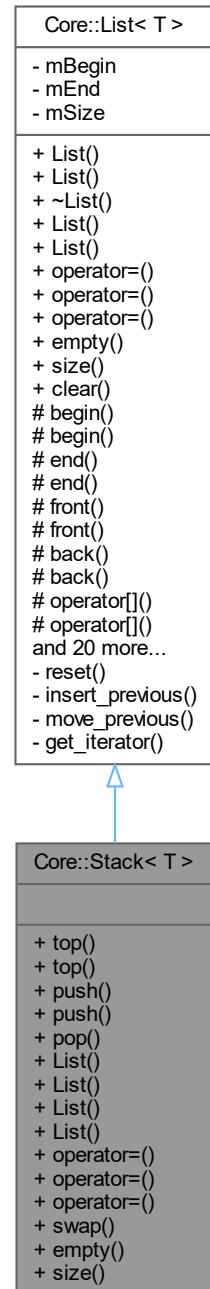
- src/Algorithms/LinkedList/Stack.hpp
- src/Algorithms/LinkedList/Stack.cpp

## 7.54 Core::Stack< T > Class Template Reference

The stack container.

```
#include <Stack.hpp>
```

Inheritance diagram for Core::Stack< T >:



## Public Member Functions

- `T & top ()`  
*Returns the reference to the top element of the stack.*
- `const T & top () const`  
*Returns the const reference to the top element of the stack.*
- `void push (const T &value)`  
*Pushes the given value to the top of the stack.*
- `void push (T &&value)`  
*Pushes the given value to the top of the stack.*
- `void pop ()`  
*Removes the top element of the stack.*
- `List ()`  
*Default constructor.*
- `List (std::initializer_list< T > list)`  
*Constructs the container with the contents of the initializer list.*
- `List (const List< T > &list)`  
*Copy constructor.*
- `List (List< T > &&list)`  
*Move constructor.*
- `List< T > & operator= (std::initializer_list< T > list)`  
*Copy assign the container with the contents of the initializer list.*
- `List< T > & operator= (const List< T > &list)`  
*Copy assignment operator.*
- `List< T > & operator= (List< T > &&list)`  
*Move assignment operator.*
- `void swap (List< T > &other)`  
*Swaps the contents of the list with those of other.*
- `bool empty () const`  
*Check whether the container is empty.*
- `std::size_t size () const`  
*Returns the size of the container.*
- `void clear ()`  
*Frees all elements in the container.*

## Protected Member Functions

- `iterator begin ()`
- `const_iterator begin () const`
- `iterator end ()`
- `const_iterator end () const`
- `T & front ()`  
*Returns the reference to the element at the front of the container.*
- `const T & front () const`  
*Returns the reference to the element at the front of the container.*
- `T & back ()`  
*Returns the reference to the element at the back of the container.*
- `const T & back () const`  
*Returns the reference to the element at the back of the container.*
- `T & operator[] (std::size_t index)`

- `const T & operator[] (std::size_t index) const`

*Returns the reference to the element at the given index.*
- `T & at (std::size_t index)`

*Returns the reference to the element at the given index.*
- `const T & at (std::size_t index) const`

*Returns the reference to the element at the given index.*
- `void push_front (const T &value)`

*Pushes the element to the front of the container.*
- `void push_back (const T &value)`

*Pushes the element to the back of the container.*
- `void pop_front ()`

*Removes the element at the front of the container.*
- `void pop_back ()`

*Removes the element at the back of the container.*
- `iterator remove (const iterator &it)`

*Removes the element iterator in the container.*
- `int remove (const T &value, const iterator &begin, const iterator &end)`

*Removes the elements in the range [begin, end]*
- `int remove (const T &value)`

*Removes the elements that has the same value as the given one.*
- `int remove_if (std::function< bool(const T &) > predicate, const iterator &begin, const iterator &end)`
- `int remove_if (std::function< bool(const T &) > predicate)`
- `void resize (std::size_t count)`

*Resizes the container to contain the given number of elements.*
- `void resize (std::size_t count, const T &value)`

*Resizes the container to contain the given number of elements.*
- `void assign (std::size_t count, const T &value)`

*Assigns new contents to the list, replacing its current content with count copies of value*
- `void assign (const const_iterator &begin, const const_iterator &end)`

*Assigns new contents to the list, replacing its current content.*
- `void assign (std::initializer_list< T > list)`

*Assigns new contents to the list, replacing its current content.*
- `std::size_t unique ()`

*Eliminates all except the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.*
- `std::size_t unique (std::function< bool(const T &, const T &) > predicate)`

*Eliminates all except the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.*
- `void reverse ()`

*Reverses the order of the elements in the list.*
- `void swap (List< T > &other)`

*Swaps the contents of the list with those of other.*

## Private Types

- using `List = Core::List< T >`

## Private Member Functions

- void `reset ()`
- void `insert_previous (const iterator &it, const iterator &it_prev)`  
*Insert an iterator before the specified iterator.*
- `iterator move_previous (const iterator &it, const iterator &first, const iterator &last)`
- `iterator get_iterator (std::size_t index)`  
*Returns an iterator to the element at index index*

## Private Attributes

- `iterator mBegin`  
*The head of the list.*
- `iterator mEnd`  
*The end of the list (one past the last element)*
- `std::size_t mSize`  
*The size of the list.*

### 7.54.1 Detailed Description

```
template<typename T>
class Core::Stack< T >
```

The stack container.

Template Parameters

<code>T</code>	the type of the elements
----------------	--------------------------

### 7.54.2 Member Typedef Documentation

#### 7.54.2.1 List

```
template<typename T >
using Core::Stack< T >::List = Core::List< T > [private]
```

### 7.54.3 Member Function Documentation

### 7.54.3.1 assign() [1/3]

```
template<typename T >
void Core::List< T >::assign (
    const const_iterator & begin,
    const const_iterator & end ) [inline], [protected], [inherited]
```

Assigns new contents to the list, replacing its current content.

#### Parameters

<i>begin</i>	The iterator to the first element to be assigned
<i>end</i>	The iterator to the last element to be assigned

### 7.54.3.2 assign() [2/3]

```
template<typename T >
void Core::List< T >::assign (
    std::initializer_list< T > list ) [inline], [protected], [inherited]
```

Assigns new contents to the list, replacing its current content.

#### Parameters

<i>list</i>	The initializer list to be assigned
-------------	-------------------------------------

### 7.54.3.3 assign() [3/3]

```
template<typename T >
void Core::List< T >::assign (
    std::size_t count,
    const T & value ) [inline], [protected], [inherited]
```

Assigns new contents to the list, replacing its current content with `count` copies of `value`

#### Parameters

<i>count</i>	The new size of the container
<i>value</i>	The value to be used to fill the container

### 7.54.3.4 at() [1/2]

```
template<typename T >
```

```
T & Core::List< T >::at (
    std::size_t index ) [inline], [protected], [inherited]
```

Returns the reference to the element at the given index.

#### Parameters

<i>index</i>	The index of the element
--------------	--------------------------

#### Returns

T& The reference to the element at the given index

#### Exceptions

<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------

std::out\_of\_range: index out of range

### 7.54.3.5 **at()** [2/2]

```
template<typename T >
const T & Core::List< T >::at (
    std::size_t index ) const [inline], [protected], [inherited]
```

Returns the reference to the element at the given index.

#### Parameters

<i>index</i>	The index of the element
--------------	--------------------------

#### Returns

T& The reference to the element at the given index

#### Exceptions

<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------

std::out\_of\_range: index out of range

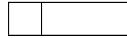
### 7.54.3.6 **back()** [1/2]

```
template<typename T >
T & Core::List< T >::back () [inline], [protected], [inherited]
```

Returns the reference to the element at the back of the container.

**Returns**

T& The reference to the element at the back of the container

**Exceptions**

undefined behavior: null pointer dereference

**7.54.3.7 back() [2/2]**

```
template<typename T >
const T & Core::List< T >::back ( ) const [inline], [protected], [inherited]
```

Returns the reference to the element at the back of the container.

**Returns**

T& The reference to the element at the back of the container

**Exceptions**

undefined behavior: null pointer dereference

**7.54.3.8 begin() [1/2]**

```
template<typename T >
iterator Core::List< T >::begin ( ) [inline], [protected], [inherited]
```

**7.54.3.9 begin() [2/2]**

```
template<typename T >
const_iterator Core::List< T >::begin ( ) const [inline], [protected], [inherited]
```

**7.54.3.10 clear()**

```
template<typename T >
void Core::List< T >::clear ( ) [inline], [inherited]
```

Frees all elements in the container.

### 7.54.3.11 `empty()`

```
template<typename T >
bool Core::List< T >::empty ( ) const [inline]
```

Check whether the container is empty.

#### Return values

<i>true</i>	The container is empty
<i>false</i>	The container is not empty

### 7.54.3.12 `end() [1/2]`

```
template<typename T >
iterator Core::List< T >::end ( ) [inline], [protected], [inherited]
```

### 7.54.3.13 `end() [2/2]`

```
template<typename T >
const_iterator Core::List< T >::end ( ) const [inline], [protected], [inherited]
```

### 7.54.3.14 `front() [1/2]`

```
template<typename T >
T & Core::List< T >::front ( ) [inline], [protected], [inherited]
```

Returns the reference to the element at the front of the container.

#### Returns

T& The reference to the element at the front of the container

#### Exceptions



undefined behavior: null pointer dereference

### 7.54.3.15 `front() [2/2]`

```
template<typename T >
```

```
const T & Core::List< T >::front ( ) const [inline], [protected], [inherited]
```

Returns the reference to the element at the front of the container.

#### Returns

T& The reference to the element at the front of the container

#### Exceptions

--	--

undefined behavior: null pointer dereference

### 7.54.3.16 get\_iterator()

```
template<typename T >
iterator Core::List< T >::get_iterator (
    std::size_t index ) [inline], [private], [inherited]
```

Returns an iterator to the element at index index

#### Parameters

<i>index</i>	The index of the element
--------------	--------------------------

#### Returns

An iterator to the element at index index

#### Exceptions

<i>std::out_of_range</i>	if `index` is out of range
--------------------------	----------------------------

### 7.54.3.17 insert\_previous()

```
template<typename T >
void Core::List< T >::insert_previous (
    const iterator & it,
    const iterator & it_prev ) [inline], [private], [inherited]
```

Insert an iterator before the specified iterator.

#### Parameters

<i>it</i>	The iterator to insert the iterator (prev) before
<i>it_prev</i>	The iterator to insert

### 7.54.3.18 List() [1/4]

```
template<typename T >
Core::List< T >::List ( ) [inline]
```

Default constructor.

### 7.54.3.19 List() [2/4]

```
template<typename T >
Core::List< T >::List (
    const List< T > & list ) [inline]
```

Copy constructor.

#### Parameters

<i>rhs</i>	The other container
------------	---------------------

### 7.54.3.20 List() [3/4]

```
template<typename T >
Core::List< T >::List (
    List< T > && list ) [inline]
```

Move constructor.

#### Parameters

<i>rhs</i>	The other container
------------	---------------------

### 7.54.3.21 List() [4/4]

```
template<typename T >
Core::List< T >::List (
    std::initializer_list< T > list ) [inline]
```

Constructs the container with the contents of the initializer list.

**Parameters**

<i>init_list</i>	The initializer list
------------------	----------------------

**7.54.3.22 move\_previous()**

```
template<typename T >
iterator Core::List< T >::move_previous (
    const iterator & it,
    const iterator & first,
    const iterator & last ) [inline], [private], [inherited]
```

**7.54.3.23 operator=() [1/3]**

```
template<typename T >
List< T > & Core::List< T >::operator= (
    const List< T > & list ) [inline]
```

Copy assignment operator.

**Parameters**

<i>rhs</i>	The other container
------------	---------------------

**7.54.3.24 operator=() [2/3]**

```
template<typename T >
List< T > & Core::List< T >::operator= (
    List< T > && list ) [inline]
```

Move assignment operator.

**Parameters**

<i>rhs</i>	The other container
------------	---------------------

**7.54.3.25 operator=() [3/3]**

```
template<typename T >
```

```
List< T > & Core::List< T >::operator= (
    std::initializer_list< T > list ) [inline]
```

Copy assign the container with the contents of the initializer list.

#### Parameters

<i>init_list</i>	The initializer list
------------------	----------------------

### 7.54.3.26 operator[]( ) [1/2]

```
template<typename T >
T & Core::List< T >::operator[] (
    std::size_t index ) [inline], [protected], [inherited]
```

Returns the reference to the element at the given index.

#### Parameters

<i>index</i>	The index of the element
--------------	--------------------------

#### Returns

T& The reference to the element at the given index

#### Exceptions



std::out\_of\_range: index out of range

### 7.54.3.27 operator[]( ) [2/2]

```
template<typename T >
const T & Core::List< T >::operator[] (
    std::size_t index ) const [inline], [protected], [inherited]
```

Returns the reference to the element at the given index.

#### Parameters

<i>index</i>	The index of the element
--------------	--------------------------

#### Returns

T& The reference to the element at the given index

**Exceptions**

<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------

std::out\_of\_range: index out of range

**7.54.3.28 pop()**

```
template<typename T >
void Core::Stack< T >::pop ( ) [inline]
```

Removes the top element of the stack.

**7.54.3.29 pop\_back()**

```
template<typename T >
void Core::List< T >::pop_back ( ) [inline], [protected], [inherited]
```

Removes the element at the back of the container.

**7.54.3.30 pop\_front()**

```
template<typename T >
void Core::List< T >::pop_front ( ) [inline], [protected], [inherited]
```

Removes the element at the front of the container.

**7.54.3.31 push() [1/2]**

```
template<typename T >
void Core::Stack< T >::push (
    const T & value ) [inline]
```

Pushes the given value to the top of the stack.

**Parameters**

<b>value</b>	the value to be pushed
--------------	------------------------

### 7.54.3.32 `push()` [2/2]

```
template<typename T >
void Core::Stack< T >::push (
    T && value )  [inline]
```

Pushes the given value to the top of the stack.

#### Parameters

<code>value</code>	the value to be pushed
--------------------	------------------------

### 7.54.3.33 `push_back()`

```
template<typename T >
void Core::List< T >::push_back (
    const T & value )  [inline], [protected], [inherited]
```

Pushes the element to the back of the container.

#### Parameters

<code>elem</code>	The element to be pushed into the back
-------------------	--

### 7.54.3.34 `push_front()`

```
template<typename T >
void Core::List< T >::push_front (
    const T & value )  [inline], [protected], [inherited]
```

Pushes the element to the front of the container.

#### Parameters

<code>elem</code>	The element to be pushed into the front
-------------------	---

### 7.54.3.35 `remove()` [1/3]

```
template<typename T >
iterator Core::List< T >::remove (
    const iterator & it )  [inline], [protected], [inherited]
```

Removes the element iterator in the container.

**Parameters**

<i>it</i>	The iterator to the element to be removed
-----------	---

**Returns**

iterator The iterator to the next element

**Exceptions**

--	--

std::out\_of\_range: iterator out of range

**7.54.3.36 remove() [2/3]**

```
template<typename T >
int Core::List< T >::remove (
    const T & value )  [inline], [protected], [inherited]
```

Removes the elements that has the same value as the given one.

**Parameters**

<i>value</i>	The value of the elements to be removed
--------------	---

**Returns**

iterator The iterator to the next element

**7.54.3.37 remove() [3/3]**

```
template<typename T >
int Core::List< T >::remove (
    const T & value,
    const iterator & begin,
    const iterator & end )  [inline], [protected], [inherited]
```

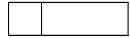
Removes the elements in the range [begin, end)

**Parameters**

<i>begin</i>	The iterator to the first element to be removed
<i>end</i>	The iterator to the last element to be removed

**Returns**

iterator The iterator to the next element

**Exceptions**

std::out\_of\_range: iterator out of range

**7.54.3.38 remove\_if() [1/2]**

```
template<typename T >
int Core::List< T >::remove_if (
    std::function< bool(const T &) > predicate ) [inline], [protected], [inherited]
```

**7.54.3.39 remove\_if() [2/2]**

```
template<typename T >
int Core::List< T >::remove_if (
    std::function< bool(const T &) > predicate,
    const iterator & begin,
    const iterator & end ) [inline], [protected], [inherited]
```

**7.54.3.40 reset()**

```
template<typename T >
void Core::List< T >::reset () [inline], [private], [inherited]
```

**7.54.3.41 resize() [1/2]**

```
template<typename T >
void Core::List< T >::resize (
    std::size_t count ) [inline], [protected], [inherited]
```

Resizes the container to contain the given number of elements.

**Parameters**

<i>count</i>	The new size of the container
--------------	-------------------------------

### 7.54.3.42 `resize()` [2/2]

```
template<typename T >
void Core::List< T >::resize (
    std::size_t count,
    const T & value ) [inline], [protected], [inherited]
```

Resizes the container to contain the given number of elements.

#### Parameters

<i>count</i>	The new size of the container
<i>value</i>	The value to be used to fill the container

### 7.54.3.43 `reverse()`

```
template<typename T >
void Core::List< T >::reverse () [inline], [protected], [inherited]
```

Reverses the order of the elements in the list.

### 7.54.3.44 `size()`

```
template<typename T >
std::size_t Core::List< T >::size () const [inline]
```

Returns the size of the container.

#### Returns

The size of the container

### 7.54.3.45 `swap()` [1/2]

```
template<typename T >
void Core::List< T >::swap (
    List< T > & other ) [inline], [protected], [inherited]
```

Swaps the contents of the list with those of other.

#### Parameters

<i>other</i>	list to exchange the contents with
--------------	------------------------------------

### 7.54.3.46 swap() [2/2]

```
template<typename T >
void Core::List< T >::swap (
    List< T > & other ) [inline]
```

Swaps the contents of the list with those of other.

#### Parameters

<i>other</i>	list to exchange the contents with
--------------	------------------------------------

### 7.54.3.47 top() [1/2]

```
template<typename T >
T & Core::Stack< T >::top ( ) [inline]
```

Returns the reference to the top element of the stack.

#### Returns

T& the reference to the top element of the stack

### 7.54.3.48 top() [2/2]

```
template<typename T >
const T & Core::Stack< T >::top ( ) const [inline]
```

Returns the const reference to the top element of the stack.

#### Returns

const T& the const reference to the top element of the stack

### 7.54.3.49 unique() [1/2]

```
template<typename T >
std::size_t Core::List< T >::unique ( ) [inline], [protected], [inherited]
```

Eliminates all except the first element from every consecutive group of equivalent elements from the range [*first*, *last*) and returns a past-the-end iterator for the new logical end of the range.

#### Returns

the resulting size

### 7.54.3.50 unique() [2/2]

```
template<typename T >
std::size_t Core::List< T >::unique (
    std::function< bool(const T &, const T &) > predicate ) [inline], [protected],
[inherited]
```

Eliminates all except the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.

#### Parameters

<i>predicate</i>	Binary predicate that, taking two values of the same type than those contained in the list, returns true to remove the element passed as first argument from the container, and false otherwise.
------------------	--

#### Returns

the resulting size

## 7.54.4 Member Data Documentation

### 7.54.4.1 mBegin

```
template<typename T >
iterator Core::List< T >::mBegin [private], [inherited]
```

The head of the list.

### 7.54.4.2 mEnd

```
template<typename T >
iterator Core::List< T >::mEnd [private], [inherited]
```

The end of the list (one past the last element)

### 7.54.4.3 mSize

```
template<typename T >
std::size_t Core::List< T >::mSize [private], [inherited]
```

The size of the list.

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

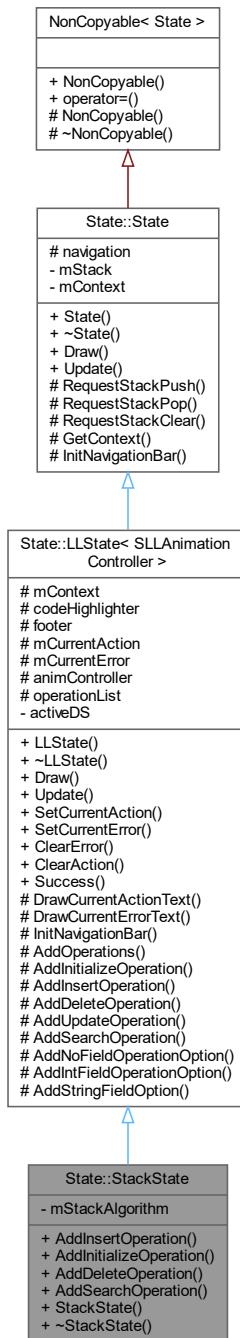
- src/Core/[Stack.hpp](#)

## 7.55 State::StackState Class Reference

The state class that is used to represent the stack state/scene of the application.

```
#include <StackState.hpp>
```

Inheritance diagram for State::StackState:



## Public Types

- `typedef std::unique_ptr< State > Ptr`

*A unique pointer to the state object.*

## Public Member Functions

- `void AddInsertOperation ()`  
*Add an insert operation to the stack algorithm.*
- `void AddInitializeOperation ()`  
*Add an initialize operation to the stack algorithm.*
- `void AddDeleteOperation ()`  
*Add an update operation to the stack algorithm.*
- `void AddSearchOperation ()`  
*Add a delete operation to the stack algorithm.*
- `StackState (StateStack &stack, Context context)`  
*Construct a new StackState object.*
- `~StackState ()`  
*Destroy the StackState object.*
- `virtual void Draw ()`  
*Draw the linked list state to the screen.*
- `virtual bool Update (float dt)`  
*Update the linked list state.*
- `virtual void SetCurrentAction (std::string action)`  
*Set the current action text.*
- `virtual void SetCurrentError (std::string error)`  
*Set the current error text.*
- `virtual void ClearError ()`  
*Clear the current error text.*
- `virtual void ClearAction ()`  
*Clear the current action text.*
- `virtual void Success ()`  
*Clear the current error and toggle the action list.*

## Protected Member Functions

- `virtual void DrawCurrentActionText ()`
- `virtual void DrawCurrentErrorText ()`
- `void InitNavigationBar ()`
- `virtual void AddOperations ()`  
*Add the operations to the operation list.*
- `virtual void AddUpdateOperation ()`  
*Add the update operation to the operation list.*
- `virtual void AddNoFieldOperationOption (GUIComponent::OperationContainer::Ptr container, std::string title, std::function< void() > action)`
- `virtual void AddIntFieldOperationOption (GUIComponent::OperationContainer::Ptr container, std::string title, Core::Deque< IntegerInput > fields, std::function< void(std::map< std::string, std::string >) > action)`
- `virtual void AddStringFieldOption (GUIComponent::OperationContainer::Ptr container, std::string title, std::string label, std::function< void(std::map< std::string, std::string >) > action)`
- `void RequestStackPush (States::ID stateID)`

- Request the state stack to push a new state/scene.
- void [RequestStackPop \(\)](#)  
Request the state stack to pop the current state/scene.
- void [RequestStackClear \(\)](#)  
Request the state stack to clear all the states/scenes.
- [Context GetContext \(\) const](#)  
Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

## Protected Attributes

- [Context mContext](#)
- [GUIComponent::CodeHighlighter::Ptr codeHighlighter](#)
- [GUIComponent::Footer< SLLAnimationController > footer](#)
- std::string [mCurrentAction](#)
- std::string [mCurrentError](#)
- T::Ptr [animController](#)
- [GUIComponent::OperationList operationList](#)
- [GUIComponent::NavigationBar navigation](#)

## Private Attributes

- [Algorithm::Stack mStackAlgorithm](#)  
*The algorithm of the stack.*
- [DataStructures::ID activeDS](#)
- [StateStack \\* mStack](#)  
*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*

### 7.55.1 Detailed Description

The state class that is used to represent the stack state/scene of the application.

#### Note

This Stack state is using the Singly Linked List [Animation](#) Controller as it can be visualized using the same animation controller.

### 7.55.2 Member Typedef Documentation

#### 7.55.2.1 Ptr

```
typedef std::unique_ptr< State > State::State::Ptr [inherited]
```

A unique pointer to the state object.

### 7.55.3 Constructor & Destructor Documentation

#### 7.55.3.1 StackState()

```
State::StackState::StackState (
    StateStack & stack,
    Context context )
```

Construct a new [StackState](#) object.

##### Parameters

<i>stack</i>	The state stack where the stack state is pushed to.
<i>context</i>	The context of the application.

#### 7.55.3.2 ~StackState()

```
State::StackState::~StackState ( )
```

Destroy the [StackState](#) object.

### 7.55.4 Member Function Documentation

#### 7.55.4.1 AddDeleteOperation()

```
void State::StackState::AddDeleteOperation ( ) [virtual]
```

Add an update operation to the stack algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.55.4.2 AddInitializeOperation()

```
void State::StackState::AddInitializeOperation ( ) [virtual]
```

Add an initialize operation to the stack algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.55.4.3 AddInsertOperation()

```
void State::StackState::AddInsertOperation ( ) [virtual]
```

Add an insert operation to the stack algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.55.4.4 AddIntFieldOperationOption()

```
void State::LLState< SLLAnimationController >::AddIntFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    Core::Deque< IntegerInput > fields,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.55.4.5 AddNoFieldOperationOption()

```
void State::LLState< SLLAnimationController >::AddNoFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::function< void() > action ) [protected], [virtual], [inherited]
```

#### 7.55.4.6 AddOperations()

```
void State::LLState< SLLAnimationController >::AddOperations [protected], [virtual], [inherited]
```

Add the operations to the operation list.

##### Note

This function should not be overridden as it calls the other Add\*Operation functions.

#### 7.55.4.7 AddSearchOperation()

```
void State::StackState::AddSearchOperation ( ) [virtual]
```

Add a delete operation to the stack algorithm.

Reimplemented from [State::LLState< SLLAnimationController >](#).

#### 7.55.4.8 AddStringFieldOption()

```
void State::LLState< SLLAnimationController >::AddStringFieldOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::string label,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.55.4.9 AddUpdateOperation()

```
void State::LLState< SLLAnimationController >::AddUpdateOperation [protected], [virtual],
[inherited]
```

Add the update operation to the operation list.

Reimplemented in [State::SLLState](#).

#### 7.55.4.10 ClearAction()

```
void State::LLState< SLLAnimationController >::ClearAction [inline], [virtual], [inherited]
```

Clear the current action text.

#### 7.55.4.11 ClearError()

```
void State::LLState< SLLAnimationController >::ClearError [inline], [virtual], [inherited]
```

Clear the current error text.

#### 7.55.4.12 Draw()

```
void State::LLState< SLLAnimationController >::Draw [inline], [virtual], [inherited]
```

Draw the linked list state to the screen.

Implements [State::State](#).

#### 7.55.4.13 DrawCurrentActionText()

```
void State::LLState< SLLAnimationController >::DrawCurrentActionText [inline], [protected],  
[virtual], [inherited]
```

#### 7.55.4.14 DrawCurrentErrorText()

```
void State::LLState< SLLAnimationController >::DrawCurrentErrorText [inline], [protected],  
[virtual], [inherited]
```

#### 7.55.4.15 GetContext()

```
State::State::Context State::State::GetContext () const [protected], [inherited]
```

Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### Returns

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### See also

[Context](#)

#### 7.55.4.16 InitNavigationBar()

```
void State::LLState< SLLAnimationController >::InitNavigationBar [protected], [inherited]
```

#### 7.55.4.17 RequestStackClear()

```
void State::State::RequestStackClear () [protected], [inherited]
```

Request the state stack to clear all the states/scenes.

##### See also

[StateStack::ClearStates](#)

#### 7.55.4.18 RequestStackPop()

```
void State::State::RequestStackPop ( ) [protected], [inherited]
```

Request the state stack to pop the current state/scene.

See also

[StateStack::PopState](#)

#### 7.55.4.19 RequestStackPush()

```
void State::State::RequestStackPush (
    States::ID stateID ) [protected], [inherited]
```

Request the state stack to push a new state/scene.

Parameters

<i>stateID</i>	The ID of the state/scene that will be pushed
----------------	---

See also

[StateStack::PushState](#)

#### 7.55.4.20 SetCurrentAction()

```
void State::LLState< SLLAnimationController >::SetCurrentAction (
    std::string action ) [inline], [virtual], [inherited]
```

Set the current action text.

Parameters

<i>action</i>	The current action text.
---------------	--------------------------

#### 7.55.4.21 SetCurrentError()

```
void State::LLState< SLLAnimationController >::SetCurrentError (
    std::string error ) [inline], [virtual], [inherited]
```

Set the current error text.

**Parameters**

<code>error</code>	The current error text.
--------------------	-------------------------

**7.55.4.22 Success()**

```
void State::LLState< SLLAnimationController >::Success [inline], [virtual], [inherited]
```

Clear the current error and toggle the action list.

**7.55.4.23 Update()**

```
bool State::LLState< SLLAnimationController >::Update (
    float dt ) [virtual], [inherited]
```

Update the linked list state.

**Parameters**

<code>dt</code>	The delta time between two frames.
-----------------	------------------------------------

**Returns**

`true` If the update was successful.  
`false` If the update was unsuccessful.

Implements [State::State](#).

**7.55.5 Member Data Documentation****7.55.5.1 activeDS**

```
DataStructures::ID State::LLState< SLLAnimationController >::activeDS [private], [inherited]
```

**7.55.5.2 animController**

```
T::Ptr State::LLState< SLLAnimationController >::animController [protected], [inherited]
```

### 7.55.5.3 codeHighlighter

```
GUIComponent::CodeHighlighter::Ptr State::LLState< SLLAnimationController >::codeHighlighter  
[protected], [inherited]
```

### 7.55.5.4 footer

```
GUIComponent::Footer< SLLAnimationController > State::LLState< SLLAnimationController ><-  
::footer [protected], [inherited]
```

### 7.55.5.5 mContext

```
Context State::LLState< SLLAnimationController >::mContext [protected], [inherited]
```

### 7.55.5.6 mCurrentAction

```
std::string State::LLState< SLLAnimationController >::mCurrentAction [protected], [inherited]
```

### 7.55.5.7 mCurrentError

```
std::string State::LLState< SLLAnimationController >::mCurrentError [protected], [inherited]
```

### 7.55.5.8 mStack

```
StateStack* State::State::mStack [private], [inherited]
```

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

### 7.55.5.9 mStackAlgorithm

```
Algorithm::Stack State::StackState::mStackAlgorithm [private]
```

The algorithm of the stack.

### 7.55.5.10 navigation

`GUIComponent::NavigationBar State::State::navigation [protected], [inherited]`

### 7.55.5.11 operationList

```
GUICOMPONENT::OperationList State::LLState< SLLAnimationController >::operationList [protected],  
[inherited]
```

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

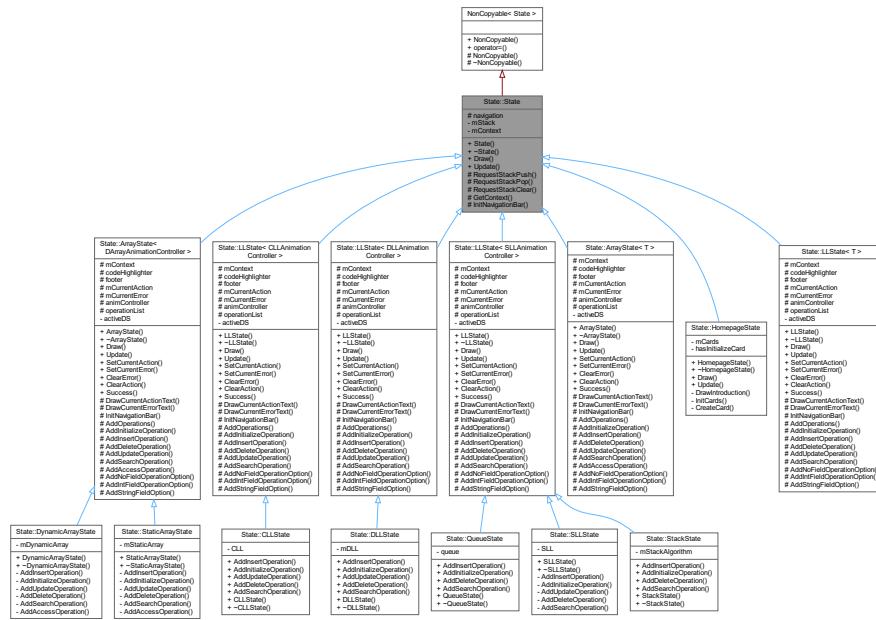
- src/States/LinkedList/StackState.hpp
  - src/States/LinkedList/StackState.cpp

## 7.56 State::State Class Reference

The base state class that is used to represent a state/scene of the application.

```
#include <State.hpp>
```

## Inheritance diagram for State::State:



## Classes

- struct Context

*The context that is used to share the resources (fonts, textures, ...) between states (scenes).*

## Public Types

- `typedef std::unique_ptr< State > Ptr`  
*A unique pointer to the state object.*

## Public Member Functions

- `State (StateStack &stack, Context context)`  
*Construct a new State object.*
- `virtual ~State ()`  
*Destroy the State object.*
- `virtual void Draw ()=0`  
*Draw the state/scene.*
- `virtual bool Update (float dt)=0`  
*Update the components of the state/scene.*

## Protected Member Functions

- `void RequestStackPush (States::ID stateID)`  
*Request the state stack to push a new state/scene.*
- `void RequestStackPop ()`  
*Request the state stack to pop the current state/scene.*
- `void RequestStackClear ()`  
*Request the state stack to clear all the states/scenes.*
- `Context GetContext () const`  
*Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).*
- `void InitNavigationBar ()`  
*Initialize the navigation bar as a component of the state/scene.*

## Protected Attributes

- `GUIComponent::NavigationBar navigation`

## Private Attributes

- `StateStack * mStack`  
*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*
- `Context mContext`  
*The context that is used to share the resources (fonts, textures, ...) between states (scenes).*

### 7.56.1 Detailed Description

The base state class that is used to represent a state/scene of the application.

## 7.56.2 Member Typedef Documentation

### 7.56.2.1 Ptr

```
typedef std::unique_ptr< State > State::State::Ptr
```

A unique pointer to the state object.

## 7.56.3 Constructor & Destructor Documentation

### 7.56.3.1 State()

```
State::State::State (
    StateStack & stack,
    Context context )
```

Construct a new [State](#) object.

#### Parameters

<code>stack</code>	The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).
<code>context</code>	The context that is used to share the resources (fonts, textures, ...) between states (scenes).

### 7.56.3.2 ~State()

```
State::State::~State ( ) [virtual]
```

Destroy the [State](#) object.

## 7.56.4 Member Function Documentation

#### 7.56.4.1 Draw()

```
virtual void State::State::Draw ( ) [pure virtual]
```

Draw the state/scene.

##### Note

This function is pure virtual and must be implemented by the derived class.

Implemented in [State::ArrayState< T >](#), [State::ArrayState< DArrayAnimationController >](#), [State::HomepageState](#), [State::LLState< T >](#), [State::LLState< CLLAnimationController >](#), [State::LLState< DLLAnimationController >](#), and [State::LLState< SLLAnimationController >](#).

#### 7.56.4.2 GetContext()

```
State::State::Context State::State::GetContext ( ) const [protected]
```

Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### Returns

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

##### See also

[Context](#)

#### 7.56.4.3 InitNavigationBar()

```
void State::State::InitNavigationBar ( ) [protected]
```

Initialize the navigation bar as a component of the state/scene.

##### See also

[GUIComponent::NavigationBar](#)

#### 7.56.4.4 RequestStackClear()

```
void State::State::RequestStackClear ( ) [protected]
```

Request the state stack to clear all the states/scenes.

##### See also

[StateStack::ClearStates](#)

#### 7.56.4.5 RequestStackPop()

```
void State::State::RequestStackPop ( ) [protected]
```

Request the state stack to pop the current state/scene.

**See also**

[StateStack::PopState](#)

#### 7.56.4.6 RequestStackPush()

```
void State::State::RequestStackPush (
    States::ID stateID ) [protected]
```

Request the state stack to push a new state/scene.

**Parameters**

<i>stateID</i>	The ID of the state/scene that will be pushed
----------------	---

**See also**

[StateStack::PushState](#)

#### 7.56.4.7 Update()

```
virtual bool State::State::Update (
    float dt ) [pure virtual]
```

Update the the components of the state/scene.

**Parameters**

<i>dt</i>	The time interval between the previous and the new frame
-----------	--

**Returns**

true (always)

**Note**

This function is pure virtual and must be implemented by the derived class.

Implemented in [State::ArrayState< T >](#), [State::ArrayState< DArrayAnimationController >](#), [State::HomepageState](#), [State::LLState< T >](#), [State::LLState< CLAnimationController >](#), [State::LLState< DLLAnimationController >](#), and [State::LLState< SLLAnimationController >](#).

### 7.56.5 Member Data Documentation

#### 7.56.5.1 mContext

```
Context State::State::mContext [private]
```

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

#### 7.56.5.2 mStack

```
StateStack* State::State::mStack [private]
```

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

#### 7.56.5.3 navigation

```
GUIComponent::NavigationBar State::State::navigation [protected]
```

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

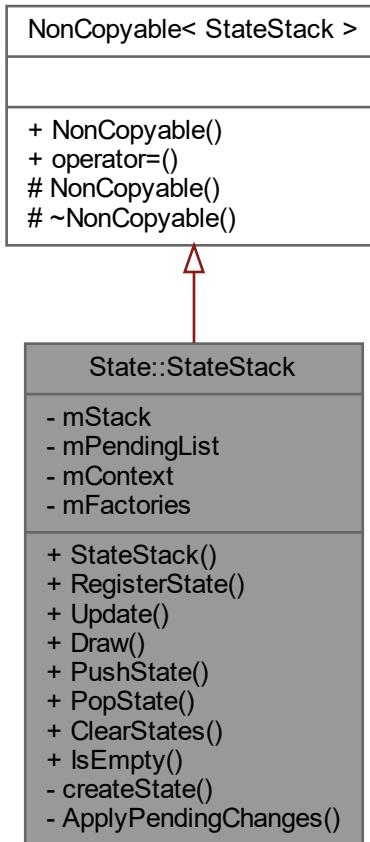
- src/[State.hpp](#)
- src/[State.cpp](#)

## 7.57 State::StateStack Class Reference

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

```
#include <StateStack.hpp>
```

Inheritance diagram for State::StateStack:



## Classes

- struct [PendingChange](#)  
*The pending changes that are applied to the state stack.*

## Public Types

- enum class [Action](#) { [Push](#) , [Pop](#) , [Clear](#) }  
*The action that is used to modify the state stack.*

## Public Member Functions

- [StateStack](#) ([State::Context](#) context)  
*Construct a new `StateStack` object.*
- template<class T >  
[void RegisterState](#) ([States::ID](#) stateID)

- **void Update (float dt)**  
*Register a new state/scene to the state stack.*
- **void Draw ()**  
*Draw the current state/scene.*
- **void PushState (States::ID stateID)**  
*Push a new state/scene to the top of the stack.*
- **void PopState ()**  
*Pop the top state/scene from the stack.*
- **void ClearStates ()**  
*Clear all the states/scenes from the stack, this is equivalent to closing the application.*
- **bool IsEmpty () const**  
*Check if the state stack is empty.*

## Private Member Functions

- **State::Ptr createState (States::ID stateID)**  
*Create a new state/scene.*
- **void ApplyPendingChanges ()**  
*Apply the pending changes to the state stack.*

## Private Attributes

- **std::vector< State::Ptr > mStack**  
*The stack that is used to store the states/scenes.*
- **Core::Deque< PendingChange > mPendingList**  
*The pending changes that are applied to the state stack.*
- **State::Context mContext**  
*The context that is used to share the resources (fonts, textures, ...) between states (scenes).*
- **std::map< States::ID, std::function< State::Ptr() > > mFactories**  
*The factories that are used to create the states/scenes.*

### 7.57.1 Detailed Description

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

### 7.57.2 Member Enumeration Documentation

#### 7.57.2.1 Action

```
enum class State::StateStack::Action [strong]
```

The action that is used to modify the state stack.

The action can be one of the following:

- Push: Add a new state to the top of the stack (or equivalent to switching to a new state/scene)
- Pop: Remove the top state from the stack (or equivalent to returning to the previous state/scene, or exiting the application if the stack is empty)
- Clear: Remove all the states from the stack (or equivalent to clearing the stack)

**Enumerator**

Push	
Pop	
Clear	

### 7.57.3 Constructor & Destructor Documentation

#### 7.57.3.1 StateStack()

```
State::StateStack::StateStack (
    State::Context context ) [explicit]
```

Construct a new [StateStack](#) object.

**Parameters**

<code>context</code>	The context that is used to share the resources (fonts, textures, ...) between states (scenes).
----------------------	---

### 7.57.4 Member Function Documentation

#### 7.57.4.1 ApplyPendingChanges()

```
void State::StateStack::ApplyPendingChanges ( ) [private]
```

Apply the pending changes to the state stack.

#### 7.57.4.2 ClearStates()

```
void State::StateStack::ClearStates ( )
```

Clear all the states/scenes from the stack, this is equivalent to closing the application.

#### 7.57.4.3 createState()

```
State::State::Ptr State::StateStack::createState (
    States::ID stateID ) [private]
```

Create a new state/scene.

**Parameters**

<code>stateID</code>	The ID of the state/scene
----------------------	---------------------------

**Returns**

A pointer to the new state/scene

**Exceptions**

<code>std::runtime_error</code>	Thrown if the state/scene ID is not registered
---------------------------------	--

**7.57.4.4 Draw()**

```
void State::StateStack::Draw ( )
```

Draw the current state/scene.

**7.57.4.5 IsEmpty()**

```
bool State::StateStack::IsEmpty ( ) const
```

Check if the state stack is empty.

**Returns**

true if the state stack is empty, false otherwise

**7.57.4.6 PopState()**

```
void State::StateStack::PopState ( )
```

Pop the top state/scene from the stack.

If the stack is empty, the application will be closed.

**7.57.4.7 PushState()**

```
void State::StateStack::PushState (   
    States::ID stateID )
```

Push a new state/scene to the top of the stack.

**Parameters**

<i>stateID</i>	The ID of the state/scene
----------------	---------------------------

**7.57.4.8 RegisterState()**

```
template<class T >
void State::StateStack::RegisterState (
    States::ID stateID )
```

Register a new state/scene to the state stack.

**Template Parameters**

<i>T</i>	The type of the state/scene
----------	-----------------------------

**Parameters**

<i>stateID</i>	The ID of the state/scene
----------------	---------------------------

**7.57.4.9 Update()**

```
void State::StateStack::Update (
    float dt )
```

Update the the components of the current state/scene.

**Parameters**

<i>dt</i>	The time interval between the previous and the new frame
-----------	--

**7.57.5 Member Data Documentation****7.57.5.1 mContext**

`State::Context State::StateStack::mContext [private]`

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

### 7.57.5.2 mFactories

```
std::map< States::ID, std::function< State::Ptr() > > State::StateStack::mFactories [private]
```

The factories that are used to create the states/scenes.

### 7.57.5.3 mPendingList

```
Core::Deque< PendingChange > State::StateStack::mPendingList [private]
```

The pending changes that are applied to the state stack.

#### See also

[PendingChange](#)

### 7.57.5.4 mStack

```
std::vector< State::Ptr > State::StateStack::mStack [private]
```

The stack that is used to store the states/scenes.

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

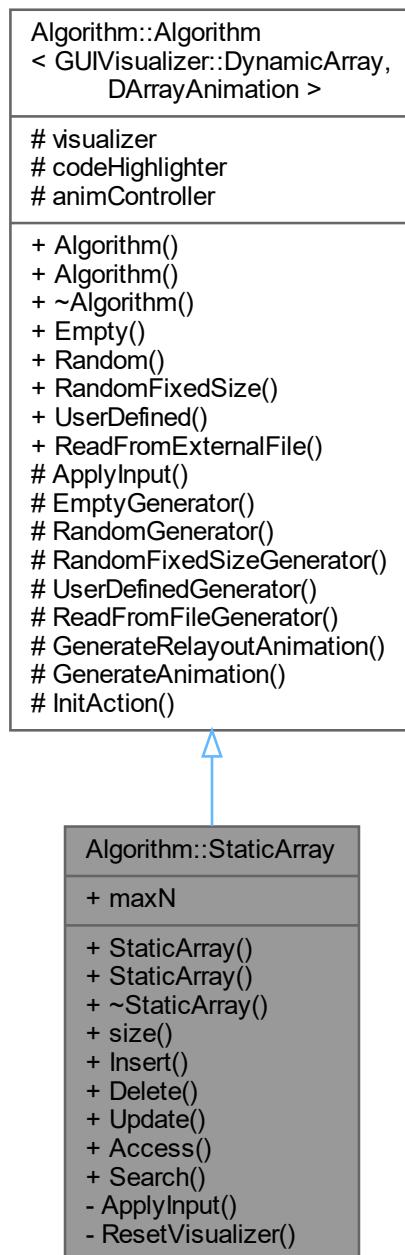
- src/[StateStack.hpp](#)
- src/[StateStack.cpp](#)

## 7.58 Algorithm::StaticArray Class Reference

The algorithm class that is used to generate step-by-step instructions for the visualization of the static array.

```
#include <StaticArray.hpp>
```

Inheritance diagram for Algorithm::StaticArray:



## Public Member Functions

- [StaticArray \(\)](#)  
*The constructor of the static array algorithm.*
- [StaticArray \(GUIComponent::CodeHighlighter::Ptr codeHighlighter, DArrayAnimationController::Ptr animController, FontHolder \\*fonts\)](#)  
*The constructor of the static array algorithm.*

- `~StaticArray ()`  
*The destructor of the static array algorithm.*
- `std::size_t size () const`  
*Return the size of the static array.*
- `void Insert (int index, int value)`  
*Push a new element to the static array.*
- `void Delete (int index)`  
*Erase an element with the given index from the static array.*
- `void Update (int index, int value)`  
*Update the value of an element with the given index.*
- `void Access (int index)`  
*Access an element with the given index.*
- `void Search (int value)`  
*Search for an element with the given value.*
- `virtual void Empty ()`  
*Initialize an empty data structure, and generate animation.*
- `virtual void Random ()`  
*Initialize a data structure with random values input, and then generate animation.*
- `virtual void RandomFixedSize (int N)`  
*Initialize a fixed size data structure with random values input, and then generate animation.*
- `virtual void UserDefined (std::string input)`  
*Initialize a data structure with input from user, and then generate animation.*
- `virtual void ReadFromExternalFile (std::string path)`  
*Initialize a data structure with input from external file (which is used to store the input), and then generate animation.*

## Static Public Attributes

- `static constexpr int maxN = 16`  
*The maximum size of the static array.*

## Protected Member Functions

- `std::vector< int > EmptyGenerator ()`  
*Generate an empty input, this is used to generate the input for `Empty ()`*
- `std::vector< int > RandomGenerator ()`  
*Generate a random input, this is used to generate the input for `Random ()`*
- `std::vector< int > RandomFixedSizeGenerator (int nSize)`  
*Generate a random input with fixed size, this is used to generate the input for `RandomFixedSize ()`*
- `std::vector< int > UserDefinedGenerator (std::string input)`  
*Generate an input from user, this is used to generate the input for `UserDefined ()`*
- `std::vector< int > ReadFromFileGenerator (std::string inputFile)`  
*Parse the input from external file, this is used to generate the input for `ReadFromExternalFile ()`*
- `virtual void GenerateRelayoutAnimation (Vector2 newPosition)`  
*Generate the relayout animation (which reposition the data structure to a new position on the screen)*
- `virtual DArrayAnimation GenerateAnimation (float duration, int highlightLine, std::string actionDescription)`  
*Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)*
- `virtual void InitAction (std::vector< std::string > code)`  
*Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.*

## Protected Attributes

- `GUIVisualizer::DynamicArray visualizer`  
*Visualizer for the algorithm (which is used to draw animation generated by the algorithm)*
- `GUICOMPONENT::CodeHighlighter::Ptr codeHighlighter`  
*Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)*
- `Animation::AnimationController< DArrayAnimation >::Ptr animController`  
*Animation controller for the algorithm (which is used to control the animation generated by the algorithm)*

## Private Member Functions

- `void ApplyInput (std::vector< int > input, std::size_t nMaxSize)`  
*Apply an input to the static array.*
- `void ResetVisualizer ()`  
*Reset the visualizer of the static array.*

### 7.58.1 Detailed Description

The algorithm class that is used to generate step-by-step instructions for the visualization of the static array.

#### Note

In this Static Array algorithm class, the visualizer I use for the static array is a dynamic array with a fixed size (and hide all the inaccesible elements). This is because the visualization of the static array and the dynamic array have a lot in common.

### 7.58.2 Constructor & Destructor Documentation

#### 7.58.2.1 StaticArray() [1/2]

```
Algorithm::StaticArray::StaticArray ( )
```

The constructor of the static array algorithm.

#### 7.58.2.2 StaticArray() [2/2]

```
Algorithm::StaticArray::StaticArray (
    GUICOMPONENT::CodeHighlighter::Ptr codeHighlighter,
    DArrayAnimationController::Ptr animController,
    FontHolder * fonts )
```

The constructor of the static array algorithm.

**Parameters**

<i>codeHighlighter</i>	The code highlighter of the static array algorithm.
<i>animController</i>	The animation controller of the static array algorithm.
<i>fonts</i>	The fonts of the static array algorithm.

**7.58.2.3 ~StaticArray()**

```
Algorithm::StaticArray::~StaticArray ( )
```

The destructor of the static array algorithm.

**Note**

This destructor is not virtual because this class is not designed to be inherited.

**7.58.3 Member Function Documentation****7.58.3.1 Access()**

```
void Algorithm::StaticArray::Access (
    int index )
```

Access an element with the given index.

**Parameters**

<i>index</i>	The index of the element to be accessed.
--------------	--

**7.58.3.2 ApplyInput()**

```
void Algorithm::StaticArray::ApplyInput (
    std::vector< int > input,
    std::size_t nMaxSize ) [private], [virtual]
```

Apply an input to the static array.

**Note**

As the Static Array algorithm use a dynamic array as its visualizer, I need to hide all the elements that are not accessible when it is initialized.

Reimplemented from [Algorithm::Algorithm< GUIVisualizer::DynamicArray, DArrayAnimation >](#).

### 7.58.3.3 Delete()

```
void Algorithm::StaticArray::Delete (
    int index )
```

Erase an element with the given index from the static array.

#### Parameters

<i>index</i>	The index of the element to be erased.
--------------	--

#### Note

The actual erasing process is done by setting the value of the element to 0, and then shifting all the elements after it to the left.

### 7.58.3.4 Empty()

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::Empty [virtual],
[inherited]
```

Initialize an empty data structure, and generate animation.

### 7.58.3.5 EmptyGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::←
EmptyGenerator [protected], [inherited]
```

Generate an empty input, this is used to generate the input for [Empty \(\)](#)

#### Returns

`std::vector< int >` the empty input

### 7.58.3.6 GenerateAnimation()

```
DArrayAnimation Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::←
GenerateAnimation (
    float duration,
    int highlightLine,
    std::string actionDescription ) [protected], [virtual], [inherited]
```

Generate an animation (which only contains the metadata of the animation, such as the duration, the highlight line, and the action description, but not the actual animation)

**Parameters**

<i>duration</i>	the duration of the animation
<i>highlightLine</i>	the line to be highlighted in the code highlighter (if the line is -1, then no line will be highlighted)
<i>actionDescription</i>	the description of the action (which is used to display in the animation)

**Returns**

AnimationState the empty animation state (which currently only contains the metadata of the animation)

**7.58.3.7 GenerateRelayoutAnimation()**

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::GenerateRelayout<=
Animation (
    Vector2 newPosition ) [inline], [protected], [virtual], [inherited]
```

Generate the relayout animation (which reposition the data structure to a new position on the screen)

**Parameters**

<i>newPosition</i>	the new position of the data structure
--------------------	--

**7.58.3.8 InitAction()**

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::InitAction (
    std::vector< std::string > code ) [protected], [virtual], [inherited]
```

Clear the animation controller and the code highlighter, act as a helper function for initialize a new instruction.

**Note**

This function is used to clear the animation controller and the code highlighter, and then initialize a new instruction

**7.58.3.9 Insert()**

```
void Algorithm::StaticArray::Insert (
    int index,
    int value )
```

Push a new element to the static array.

**Parameters**

<code>value</code>	The value of the new element.
--------------------	-------------------------------

**Note**

The new element can only be pushed to the static array if the static array is not full (which I will check if the last element's value is 0).

**7.58.3.10 Random()**

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::Random [virtual], [inherited]
```

Initialize a data structure with random values input, and then generate animation.

**7.58.3.11 RandomFixedSize()**

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::RandomFixedSize ( int N ) [virtual], [inherited]
```

Initialize a fixed size data structure with random values input, and then generate animation.

**Parameters**

<code>N</code>	the size of the data structure
----------------	--------------------------------

**7.58.3.12 RandomFixedSizeGenerator()**

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::RandomFixedSizeGenerator ( int nSize ) [protected], [inherited]
```

Generate a random input with fixed size, this is used to generate the input for `RandomFixedSize()`

**Parameters**

<code>nSize</code>	the size of the input
--------------------	-----------------------

**Returns**

std::vector< int > the random input with fixed size

**7.58.3.13 RandomGenerator()**

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::←
RandomGenerator [protected], [inherited]
```

Generate a random input, this is used to generate the input for [Random\(\)](#)

**Returns**

std::vector< int > the random input

**7.58.3.14 ReadFromExternalFile()**

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::ReadFromExternal←
File (
    std::string path ) [virtual], [inherited]
```

Initialize a data structure with input from external file (which is used to store the input), and then generate animation.

**Parameters**

<i>path</i>	the path to the external file
-------------	-------------------------------

**7.58.3.15 ReadFromFileGenerator()**

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::←
ReadFromFileGenerator (
    std::string inputFile ) [protected], [inherited]
```

Parse the input from external file, this is used to generate the input for [ReadFromExternalFile\(\)](#)

**Parameters**

<i>inputFile</i>	the path to the external file
------------------	-------------------------------

**Returns**

std::vector< int > the input from external file

### 7.58.3.16 ResetVisualizer()

```
void Algorithm::StaticArray::ResetVisualizer ( ) [private]
```

Reset the visualizer of the static array.

#### Note

This function is used to reset all of the effects that are only used to visualize the algorithm (i.e highlight the elements that are currently iterated).

### 7.58.3.17 Search()

```
void Algorithm::StaticArray::Search (
    int value )
```

Search for an element with the given value.

#### Parameters

<i>value</i>	The value of the element to be searched.
--------------	--

### 7.58.3.18 size()

```
std::size_t Algorithm::StaticArray::size ( ) const
```

Return the size of the static array.

#### Returns

The size of the static array.

### 7.58.3.19 Update()

```
void Algorithm::StaticArray::Update (
    int index,
    int value )
```

Update the value of an element with the given index.

#### Parameters

<i>index</i>	The index of the element to be updated.
<i>value</i>	The new value of the element.

### 7.58.3.20 UserDefined()

```
void Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::UserDefined (
    std::string input ) [virtual], [inherited]
```

Initialize a data structure with input from user, and then generate animation.

#### Parameters

<i>input</i>	the input from user
--------------	---------------------

### 7.58.3.21 UserDefinedGenerator()

```
std::vector< int > Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::UserDefinedGenerator (
    std::string input ) [protected], [inherited]
```

Generate an input from user, this is used to generate the input for [UserDefined\(\)](#)

#### Parameters

<i>input</i>	the input from user
--------------	---------------------

#### Returns

`std::vector< int >` the input from user

## 7.58.4 Member Data Documentation

### 7.58.4.1 animController

```
Animation::AnimationController<DArrayAnimation >::Ptr Algorithm::Algorithm< GUIVisualizer::DynamicArray ,
, DArrayAnimation >::animController [protected], [inherited]
```

[Animation](#) controller for the algorithm (which is used to control the animation generated by the algorithm)

#### Note

This [Algorithm](#) class is mainly use the animation controller to add animation for the algorithm

#### 7.58.4.2 codeHighlighter

```
GUIComponent::CodeHighlighter::Ptr Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::codeHighlighter [protected], [inherited]
```

Code highlighter for the algorithm (which is used to highlight the currently running line code in the algorithm)

#### 7.58.4.3 maxN

```
constexpr int Algorithm::StaticArray::maxN = 16 [static], [constexpr]
```

The maximum size of the static array.

#### 7.58.4.4 visualizer

```
GUIVisualizer::DynamicArray Algorithm::Algorithm< GUIVisualizer::DynamicArray , DArrayAnimation >::visualizer [protected], [inherited]
```

Visualizer for the algorithm (which is used to draw animation generated by the algorithm)

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

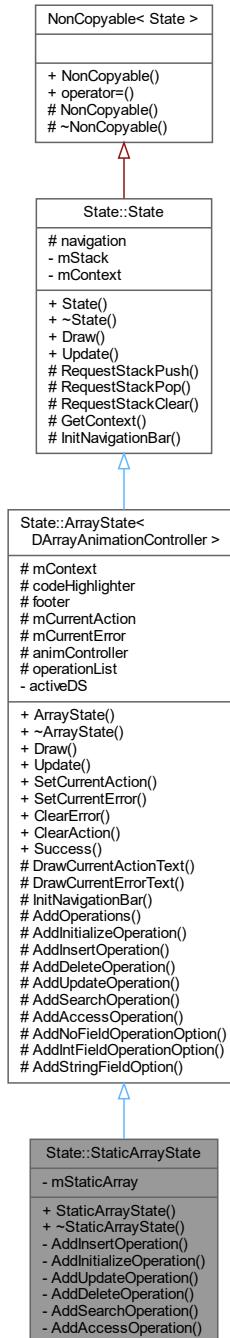
- src/Algorithms/Array/[StaticArray.hpp](#)
- src/Algorithms/Array/[StaticArray.cpp](#)

## 7.59 State::StaticArrayState Class Reference

The state class that is used to represent the static array state/scene of the application.

```
#include <StaticArrayState.hpp>
```

Inheritance diagram for State::StaticArrayState:



## Public Types

- `typedef std::unique_ptr< State > Ptr`
- A unique pointer to the state object.*

## Public Member Functions

- **StaticArrayState** (*StateStack &stack, Context context*)  
*Construct a new `StaticArrayState` object.*
- **~StaticArrayState** ()  
*Destroy the `StaticArrayState` object.*
- **virtual void Draw** ()  
*Draw the array state to the screen.*
- **virtual bool Update** (float *dt*)  
*Update the array state.*
- **virtual void SetCurrentAction** (std::string *action*)  
*Set the current action text.*
- **virtual void SetCurrentError** (std::string *error*)  
*Set the current error text.*
- **virtual void ClearError** ()  
*Clear the current error text.*
- **virtual void ClearAction** ()  
*Clear the current action text.*
- **virtual void Success** ()  
*Clear the current error and toggle the action list.*

## Protected Member Functions

- **virtual void DrawCurrentActionText** ()
- **virtual void DrawCurrentErrorText** ()
- **void InitNavigationBar** ()
- **virtual void AddOperations** ()  
*Add the operations to the operation list.*
- **virtual void AddNoFieldOperationOption** (GUIComponent::OperationContainer::Ptr *container*, std::string *title*, std::function< void() > *action*)
- **virtual void AddIntegerFieldOption** (GUIComponent::OperationContainer::Ptr *container*, std::string *title*, Core::Deque< IntegerInput > *fields*, std::function< void(std::map< std::string, std::string >) > *action*)
- **virtual void AddStringFieldOption** (GUIComponent::OperationContainer::Ptr *container*, std::string *title*, std::string *label*, std::function< void(std::map< std::string, std::string >) > *action*)
- **void RequestStackPush** (States::ID *stateID*)  
*Request the state stack to push a new state/scene.*
- **void RequestStackPop** ()  
*Request the state stack to pop the current state/scene.*
- **void RequestStackClear** ()  
*Request the state stack to clear all the states/scenes.*
- **Context GetContext** () const  
*Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).*

## Protected Attributes

- **Context mContext**
- **GUIComponent::CodeHighlighter::Ptr codeHighlighter**
- **GUIComponent::Footer< DArrayAnimationController > footer**
- **std::string mCurrentAction**
- **std::string mCurrentError**
- **T::Ptr animController**
- **GUIComponent::OperationList operationList**
- **GUIComponent::NavigationBar navigation**

## Private Member Functions

- void [AddInsertOperation \(\)](#)  
*Add an insert operation to the static array algorithm.*
- void [AddInitializeOperation \(\)](#)  
*Add an initialize operation to the static array algorithm.*
- void [AddUpdateOperation \(\)](#)  
*Add an update operation to the static array algorithm.*
- void [AddDeleteOperation \(\)](#)  
*Add a delete operation to the static array algorithm.*
- void [AddSearchOperation \(\)](#)  
*Add a search operation to the static array algorithm.*
- void [AddAccessOperation \(\)](#)  
*Add an access operation to the static array algorithm.*

## Private Attributes

- [Algorithm::StaticArray mStaticArray](#)  
*The algorithm of the static array.*
- [DataStructures::ID activeDS](#)
- [StateStack \\* mStack](#)  
*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*

### 7.59.1 Detailed Description

The state class that is used to represent the static array state/scene of the application.

#### Note

This Static Array state is using the Dynamic Array [Animation Controller](#) as it can be visualized using the same animation controller (by hiding all the inaccessible element/reserved element in dynamic array).

### 7.59.2 Member Typedef Documentation

#### 7.59.2.1 Ptr

```
typedef std::unique_ptr< State > State::State::Ptr [inherited]
```

A unique pointer to the state object.

### 7.59.3 Constructor & Destructor Documentation

#### 7.59.3.1 StaticArrayState()

```
State::StaticArrayState::StaticArrayState (
    StateStack & stack,
    Context context )
```

Construct a new [StaticArrayState](#) object.

**Parameters**

<i>stack</i>	The state stack where the static array state is pushed to.
<i>context</i>	The context of the application.

**7.59.3.2 ~StaticArrayState()**

```
State::StaticArrayState::~StaticArrayState ( )
```

Destroy the [StaticArrayState](#) object.

**7.59.4 Member Function Documentation****7.59.4.1 AddAccessOperation()**

```
void State::StaticArrayState::AddAccessOperation ( ) [private], [virtual]
```

Add an access operation to the static array algorithm.

Reimplemented from [State::ArrayState< DArrayAnimationController >](#).

**7.59.4.2 AddDeleteOperation()**

```
void State::StaticArrayState::AddDeleteOperation ( ) [private], [virtual]
```

Add a delete operation to the static array algorithm.

Reimplemented from [State::ArrayState< DArrayAnimationController >](#).

**7.59.4.3 AddInitializeOperation()**

```
void State::StaticArrayState::AddInitializeOperation ( ) [private], [virtual]
```

Add an initialize operation to the static array algorithm.

Reimplemented from [State::ArrayState< DArrayAnimationController >](#).

#### 7.59.4.4 AddInsertOperation()

```
void State::StaticArrayState::AddInsertOperation ( ) [private], [virtual]
```

Add an insert operation to the static array algorithm.

Reimplemented from [State::ArrayState< DArrayAnimationController >](#).

#### 7.59.4.5 AddIntFieldOperationOption()

```
void State::ArrayState< DArrayAnimationController >::AddIntFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    Core::Deque< IntegerInput > fields,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.59.4.6 AddNoFieldOperationOption()

```
void State::ArrayState< DArrayAnimationController >::AddNoFieldOperationOption (
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::function< void() > action ) [protected], [virtual], [inherited]
```

#### 7.59.4.7 AddOperations()

```
void State::ArrayState< DArrayAnimationController >::AddOperations [protected], [virtual],
[inherited]
```

Add the operations to the operation list.

##### Note

This function should not be overridden as it calls the other Add\*Operation functions.

#### 7.59.4.8 AddSearchOperation()

```
void State::StaticArrayState::AddSearchOperation ( ) [private], [virtual]
```

Add a search operation to the static array algorithm.

Reimplemented from [State::ArrayState< DArrayAnimationController >](#).

#### 7.59.4.9 AddStringFieldOption()

```
void State::ArrayState< DArrayAnimationController >::AddStringFieldOption ( 
    GUIComponent::OperationContainer::Ptr container,
    std::string title,
    std::string label,
    std::function< void(std::map< std::string, std::string >) > action ) [protected],
[virtual], [inherited]
```

#### 7.59.4.10 AddUpdateOperation()

```
void State::StaticArrayState::AddUpdateOperation ( ) [private], [virtual]
```

Add an update operation to the static array algorithm.

Reimplemented from [State::ArrayState< DArrayAnimationController >](#).

#### 7.59.4.11 ClearAction()

```
void State::ArrayState< DArrayAnimationController >::ClearAction [inline], [virtual], [inherited]
```

Clear the current action text.

#### 7.59.4.12 ClearError()

```
void State::ArrayState< DArrayAnimationController >::ClearError [inline], [virtual], [inherited]
```

Clear the current error text.

#### 7.59.4.13 Draw()

```
void State::ArrayState< DArrayAnimationController >::Draw [inline], [virtual], [inherited]
```

Draw the array state to the screen.

Implements [State::State](#).

**7.59.4.14 DrawCurrentActionText()**

```
void State::ArrayState< DArrayAnimationController >::DrawCurrentActionText [inline], [protected],  
[virtual], [inherited]
```

**7.59.4.15 DrawCurrentErrorText()**

```
void State::ArrayState< DArrayAnimationController >::DrawCurrentErrorText [inline], [protected],  
[virtual], [inherited]
```

**7.59.4.16 GetContext()**

```
State::State::Context State::State::GetContext () const [protected], [inherited]
```

Get the context that is used to share the resources (fonts, textures, ...) between states (scenes).

**Returns**

The context that is used to share the resources (fonts, textures, ...) between states (scenes).

**See also**

[Context](#)

**7.59.4.17 InitNavigationBar()**

```
void State::ArrayState< DArrayAnimationController >::InitNavigationBar [protected], [inherited]
```

**7.59.4.18 RequestStackClear()**

```
void State::State::RequestStackClear () [protected], [inherited]
```

Request the state stack to clear all the states/scenes.

**See also**

[StateStack::ClearStates](#)

#### 7.59.4.19 RequestStackPop()

```
void State::State::RequestStackPop ( ) [protected], [inherited]
```

Request the state stack to pop the current state/scene.

See also

[StateStack::PopState](#)

#### 7.59.4.20 RequestStackPush()

```
void State::State::RequestStackPush (
    States::ID stateID ) [protected], [inherited]
```

Request the state stack to push a new state/scene.

Parameters

<i>stateID</i>	The ID of the state/scene that will be pushed
----------------	---

See also

[StateStack::PushState](#)

#### 7.59.4.21 SetCurrentAction()

```
void State::ArrayState< DArrayAnimationController >::SetCurrentAction (
    std::string action ) [inline], [virtual], [inherited]
```

Set the current action text.

Parameters

<i>action</i>	The current action text.
---------------	--------------------------

#### 7.59.4.22 SetCurrentError()

```
void State::ArrayState< DArrayAnimationController >::SetCurrentError (
    std::string error ) [inline], [virtual], [inherited]
```

Set the current error text.

**Parameters**

<code>error</code>	The current error text.
--------------------	-------------------------

**7.59.4.23 Success()**

```
void State::ArrayState< DArrayAnimationController >::Success [inline], [virtual], [inherited]
```

Clear the current error and toggle the action list.

**7.59.4.24 Update()**

```
bool State::ArrayState< DArrayAnimationController >::Update (
    float dt ) [virtual], [inherited]
```

Update the array state.

**Parameters**

<code>dt</code>	The delta time between the previous and the current frame.
-----------------	--

**Returns**

`true` If the update is successful.  
`false` If the update is unsuccessful.

Implements [State::State](#).

**7.59.5 Member Data Documentation****7.59.5.1 activeDS**

```
DataStructures::ID State::ArrayState< DArrayAnimationController >::activeDS [private], [inherited]
```

**7.59.5.2 animController**

```
T::Ptr State::ArrayState< DArrayAnimationController >::animController [protected], [inherited]
```

### 7.59.5.3 codeHighlighter

```
GUIComponent::CodeHighlighter::Ptr State::ArrayState< DArrayAnimationController >::code←  
Highlighter [protected], [inherited]
```

### 7.59.5.4 footer

```
GUIComponent::Footer< DArrayAnimationController > State::ArrayState< DArrayAnimationController  
>::footer [protected], [inherited]
```

### 7.59.5.5 mContext

```
Context State::ArrayState< DArrayAnimationController >::mContext [protected], [inherited]
```

### 7.59.5.6 mCurrentAction

```
std::string State::ArrayState< DArrayAnimationController >::mCurrentAction [protected], [inherited]
```

### 7.59.5.7 mCurrentError

```
std::string State::ArrayState< DArrayAnimationController >::mCurrentError [protected], [inherited]
```

### 7.59.5.8 mStack

```
StateStack* State::State::mStack [private], [inherited]
```

The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).

### 7.59.5.9 mStaticArray

```
Algorithm::StaticArray State::StaticArrayState::mStaticArray [private]
```

The algorithm of the static array.

### 7.59.5.10 navigation

```
GUIComponent::NavigationBar State::State::navigation [protected], [inherited]
```

### 7.59.5.11 operationList

```
GUIComponent::OperationList State::ArrayState< DArrayAnimationController >::operationList  
[protected], [inherited]
```

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

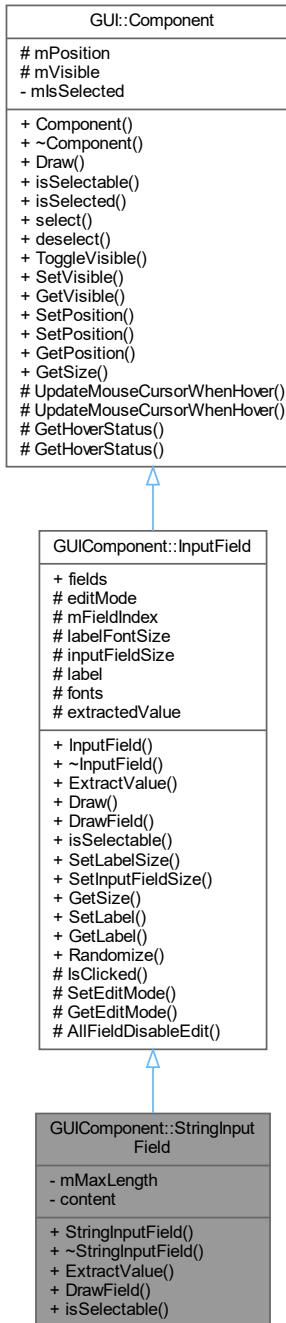
- src/States/Array/[StaticArrayState.hpp](#)
- src/States/Array/[StaticArrayState.cpp](#)

## 7.60 GUIComponent::StringInputField Class Reference

The string input field class that is used to represent a string input field in the [GUI](#).

```
#include <StringInputField.hpp>
```

Inheritance diagram for GUIComponent::StringInputField:



## Public Types

- `typedef std::shared_ptr< StringInputField > Ptr`

## Public Member Functions

- `StringInputField (FontHolder *fonts)`

- `~StringInputField ()`
- `std::string ExtractValue ()`
- `void DrawField (Vector2 base=(Vector2){0, 0})`
- `bool IsSelectable () const`  
`Check if the component is selectable.`
- `virtual void Draw (Vector2 base=(Vector2){0, 0})`  
`Draw the component.`
- `virtual void SetLabelSize (float fontSize)`
- `virtual void SetInputFieldSize (Vector2 size)`
- `virtual Vector2 GetSize ()`  
`Get the size of the component.`
- `virtual void SetLabel (std::string labelContent)`
- `virtual std::string GetLabel () const`
- `virtual void Randomize ()`
- `bool IsSelected () const`  
`Check if the component is selected.`
- `virtual void Select ()`  
`Select the component.`
- `virtual void Deselect ()`  
`Deselect the component.`
- `virtual void ToggleVisible ()`  
`Toggle the visibility of the component.`
- `virtual void SetVisible (bool visible)`  
`Set the visibility of the component.`
- `virtual bool GetVisible ()`  
`Get the visibility of the component.`
- `void SetPosition (float x, float y)`  
`Set the position of the component.`
- `void SetPosition (Vector2 position)`  
`Set the position of the component.`
- `Vector2 GetPosition ()`  
`Get the position of the component.`

## Static Public Attributes

- `static std::vector< bool > fields`

## Protected Member Functions

- `virtual bool IsClicked (Vector2 base=(Vector2){0, 0}) const`
- `virtual void SetEditMode (bool canEdit)`
- `virtual bool GetEditMode () const`
- `virtual void AllFieldDisableEdit ()`
- `virtual void UpdateMouseCursorWhenHover (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
`Update the mouse cursor when hovering.`
- `virtual void UpdateMouseCursorWhenHover (Rectangle bound, bool hover, bool noHover)`  
`Update the mouse cursor when hovering.`
- `virtual bool GetHoverStatus (std::map< std::string, Rectangle > bounds, bool hover, bool noHover)`  
`Get the hover status of the component.`
- `virtual bool GetHoverStatus (Rectangle bound, bool hover, bool noHover)`  
`Get the hover status of the component.`

## Protected Attributes

- bool `editMode`
- std::size\_t `mFieldIndex`
- float `labelFontSize`
- Vector2 `inputFieldSize`
- std::string `label`
- `FontHolder` \* `fonts`
- std::string `extractedValue`
- Vector2 `mPosition`  
*The position of the component.*
- bool `mVisible`  
*The visibility of the component.*

## Private Attributes

- std::size\_t `mMaxLength`
- std::string `content`
- bool `mIsSelected`  
*The selected status of the component.*

### 7.60.1 Detailed Description

The string input field class that is used to represent a string input field in the [GUI](#).

The string input field is used to get string input from the user.

### 7.60.2 Member Typedef Documentation

#### 7.60.2.1 Ptr

```
typedef std::shared_ptr< StringInputField > GUIComponent::StringInputField::Ptr
```

### 7.60.3 Constructor & Destructor Documentation

#### 7.60.3.1 StringInputField()

```
GUIComponent::StringInputField::StringInputField (
    FontHolder * fonts )
```

### 7.60.3.2 ~StringInputField()

```
GUIComponent::StringInputField::~StringInputField ( )
```

## 7.60.4 Member Function Documentation

### 7.60.4.1 AllFieldDisableEdit()

```
void GUIComponent::InputField::AllFieldDisableEdit ( ) [protected], [virtual], [inherited]
```

### 7.60.4.2 deselect()

```
void GUI::Component::deselect ( ) [virtual], [inherited]
```

Deselect the component.

**Deprecated** This function is deprecated.

This function deselects the component.

### 7.60.4.3 Draw()

```
void GUIComponent::InputField::Draw ( 
    Vector2 base = (Vector2){0, 0} ) [virtual], [inherited]
```

Draw the component.

This function draws the component.

#### Parameters

<i>base</i>	The base position of the component.
-------------	-------------------------------------

Reimplemented from [GUI::Component](#).

### 7.60.4.4 DrawField()

```
void GUIComponent::StringInputField::DrawField ( 
    Vector2 base = (Vector2){0, 0} ) [virtual]
```

Implements [GUIComponent::InputField](#).

#### 7.60.4.5 ExtractValue()

```
std::string GUIComponent::StringInputField::ExtractValue () [virtual]
```

Implements [GUIComponent::InputField](#).

#### 7.60.4.6 GetEditMode()

```
bool GUIComponent::InputField::GetEditMode () const [protected], [virtual], [inherited]
```

#### 7.60.4.7 GetHoverStatus() [1/2]

```
bool GUI::Component::GetHoverStatus (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.60.4.8 GetHoverStatus() [2/2]

```
bool GUI::Component::GetHoverStatus (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Get the hover status of the component.

This function returns the hover status of the component.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

#### 7.60.4.9 GetLabel()

```
std::string GUIComponent::InputField::GetLabel () const [virtual], [inherited]
```

#### 7.60.4.10GetPosition()

```
Vector2 GUI::Component::GetPosition () [inherited]
```

Get the position of the component.

This function gets the position of the component.

##### Returns

The position of the component.

#### 7.60.4.11GetSize()

```
Vector2 GUIComponent::InputField::GetSize () [virtual], [inherited]
```

Get the size of the component.

This function gets the size of the component.

##### Returns

The size of the component.

Reimplemented from [GUI::Component](#).

#### 7.60.4.12GetVisible()

```
bool GUI::Component::GetVisible () [virtual], [inherited]
```

Get the visibility of the component.

This function gets the visibility of the component.

##### Returns

The visibility of the component.

#### 7.60.4.13 IsClicked()

```
bool GUIComponent::InputField::IsClicked (
    Vector2 base = (Vector2){0, 0} ) const [protected], [virtual], [inherited]
```

#### 7.60.4.14 isSelectable()

```
bool GUIComponent::StringInputField::isSelectable () const [virtual]
```

Check if the component is selectable.

**Deprecated** This function is deprecated.

This function checks if the component is selectable.

##### Returns

True if the component is selectable.

Reimplemented from [GUIComponent::InputField](#).

#### 7.60.4.15 isSelected()

```
bool GUI::Component::isSelected () const [inherited]
```

Check if the component is selected.

**Deprecated** This function is deprecated.

This function checks if the component is selected.

##### Returns

True if the component is selected.

#### 7.60.4.16 Randomize()

```
void GUIComponent::InputField::Randomize () [virtual], [inherited]
```

Reimplemented in [GUIComponent::IntegerInputField](#).

#### 7.60.4.17 select()

```
void GUI::Component::select ( ) [virtual], [inherited]
```

Select the component.

**Deprecated** This function is deprecated.

This function selects the component.

#### 7.60.4.18 SetEditMode()

```
void GUIComponent::InputField::SetEditMode (
    bool canEdit ) [protected], [virtual], [inherited]
```

#### 7.60.4.19 SetInputFieldSize()

```
void GUIComponent::InputField::SetInputFieldSize (
    Vector2 size ) [virtual], [inherited]
```

#### 7.60.4.20 SetLabel()

```
void GUIComponent::InputField::SetLabel (
    std::string labelText ) [virtual], [inherited]
```

#### 7.60.4.21 SetLabelSize()

```
void GUIComponent::InputField::SetLabelSize (
    float fontSize ) [virtual], [inherited]
```

#### 7.60.4.22 SetPosition() [1/2]

```
void GUI::Component::SetPosition (
    float x,
    float y ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>x</i>	The x coordinate of the position.
<i>y</i>	The y coordinate of the position.

**7.60.4.23 SetPosition() [2/2]**

```
void GUI::Component::SetPosition (
    Vector2 position ) [inherited]
```

Set the position of the component.

This function sets the position of the component.

**Parameters**

<i>position</i>	The position of the component.
-----------------	--------------------------------

**7.60.4.24 SetVisible()**

```
void GUI::Component::SetVisible (
    bool visible ) [virtual], [inherited]
```

Set the visibility of the component.

This function sets the visibility of the component.

**Parameters**

<i>visible</i>	The visibility of the component.
----------------	----------------------------------

Reimplemented in [GUIComponent::OperationContainer](#), and [GUIComponent::OptionInputField](#).

**7.60.4.25 ToggleVisible()**

```
void GUI::Component::ToggleVisible ( ) [virtual], [inherited]
```

Toggle the visibility of the component.

This function toggles the visibility of the component.

#### 7.60.4.26 UpdateMouseCursorWhenHover() [1/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    Rectangle bound,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bound</i>	The bound of the component.
--------------	-----------------------------

#### 7.60.4.27 UpdateMouseCursorWhenHover() [2/2]

```
void GUI::Component::UpdateMouseCursorWhenHover (
    std::map< std::string, Rectangle > bounds,
    bool hover,
    bool noHover ) [protected], [virtual], [inherited]
```

Update the mouse cursor when hovering.

This function updates the mouse cursor when hovering.

##### Parameters

<i>bounds</i>	The list of bound of the component.
---------------	-------------------------------------

### 7.60.5 Member Data Documentation

#### 7.60.5.1 content

```
std::string GUIComponent::StringInputField::content [private]
```

#### 7.60.5.2 editMode

```
bool GUIComponent::InputField::editMode [protected], [inherited]
```

### 7.60.5.3 extractedValue

```
std::string GUIComponent::InputField::extractedValue [protected], [inherited]
```

### 7.60.5.4 fields

```
std::vector< bool > GUIComponent::InputField::fields [static], [inherited]
```

### 7.60.5.5 fonts

```
FontHolder* GUIComponent::InputField::fonts [protected], [inherited]
```

### 7.60.5.6 inputFieldSize

```
Vector2 GUIComponent::InputField::inputFieldSize [protected], [inherited]
```

### 7.60.5.7 label

```
std::string GUIComponent::InputField::label [protected], [inherited]
```

### 7.60.5.8 labelFontSize

```
float GUIComponent::InputField::labelFontSize [protected], [inherited]
```

### 7.60.5.9 mFieldIndex

```
std::size_t GUIComponent::InputField::mFieldIndex [protected], [inherited]
```

### 7.60.5.10 mIsSelected

```
bool GUI::Component::mIsSelected [private], [inherited]
```

The selected status of the component.

**Deprecated** This variable is deprecated.

This variable stores the selected status of the component.

### 7.60.5.11 mMaxLength

```
std::size_t GUIComponent::StringInputField::mMaxLength [private]
```

### 7.60.5.12 mPosition

```
Vector2 GUI::Component::mPosition [protected], [inherited]
```

The position of the component.

This variable stores the position of the component.

### 7.60.5.13 mVisible

```
bool GUI::Component::mVisible [protected], [inherited]
```

The visibility of the component.

This variable stores the visibility of the component.

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

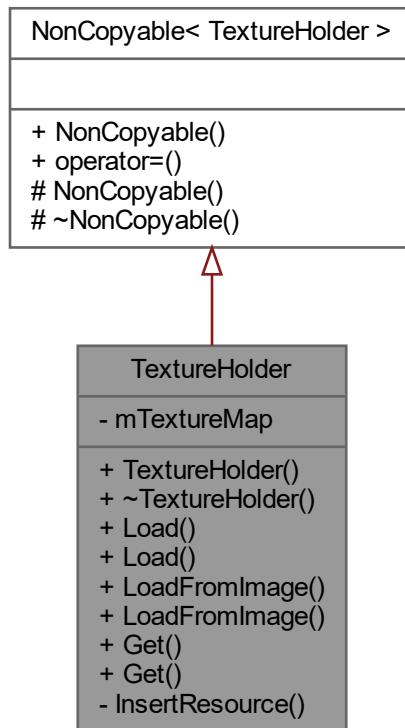
- src/Components/Common/[StringInputField.hpp](#)
- src/Components/Common/[StringInputField.cpp](#)

## 7.61 TextureHolder Class Reference

The image holder class that is used to store the images.

```
#include <TextureHolder.hpp>
```

Inheritance diagram for TextureHolder:



### Public Member Functions

- [TextureHolder \(\)](#)  
*The constructor.*
- [~TextureHolder \(\)](#)  
*The destructor.*
- [void Load \(Textures::ID id, const std::string &filename\)](#)  
*Load the image from the file.*
- [void Load \(Textures::ID id, const std::string &filename, int width, int height, bool cropCenter=false\)](#)  
*Load the texture from the file.*
- [void LoadFromImage \(Textures::ID id, Image &image\)](#)  
*Load the texture from the image.*
- [void LoadFromImage \(Textures::ID id, Image &image, int width, int height\)](#)  
*Load the texture from the image and crop it to the specified width and height.*
- [Texture & Get \(Textures::ID id\)](#)  
*Get the texture.*
- [const Texture & Get \(Textures::ID id\) const](#)  
*Get the texture.*

## Private Member Functions

- void [InsertResource \(Textures::ID id, std::unique\\_ptr< Texture > texture\)](#)  
*Insert the texture to the map.*

## Private Attributes

- std::map< [Textures::ID](#), std::unique\_ptr< Texture > > [mTextureMap](#)  
*The texture map.*

### 7.61.1 Detailed Description

The image holder class that is used to store the images.

The image holder class is used to store the images. Also, this class is using the flyweight pattern to store the images.

See also

[NonCopyable](#)

[Image](#)

### 7.61.2 Constructor & Destructor Documentation

#### 7.61.2.1 TextureHolder()

`TextureHolder::TextureHolder ( )`

The constructor.

The constructor that is used to initialize the image holder.

#### 7.61.2.2 ~TextureHolder()

`TextureHolder::~TextureHolder ( )`

The destructor.

The destructor that is used to destroy the image holder.

### 7.61.3 Member Function Documentation

#### 7.61.3.1 Get() [1/2]

`Texture & TextureHolder::Get (`  
`Textures::ID id )`

Get the texture.

This function is used to retrieve the texture.

**Parameters**

<i>id</i>	The image ID.
-----------	---------------

**Returns**

The image.

**See also**

[Texture::ID](#)

**7.61.3.2 Get() [2/2]**

```
const Texture & TextureHolder::Get (
    Textures::ID id ) const
```

Get the texture.

This function is used to retrieve the texture.

**Parameters**

<i>id</i>	The image ID.
-----------	---------------

**Returns**

The image.

**See also**

[Texture::ID](#)

**7.61.3.3 InsertResource()**

```
void TextureHolder::InsertResource (
    Textures::ID id,
    std::unique_ptr< Texture > texture ) [private]
```

Insert the texture to the map.

This function is used to insert the texture to map.

**Parameters**

<i>id</i>	The image ID.
<i>texture</i>	The texture.

**See also**[Texture::ID](#)**7.61.3.4 Load() [1/2]**

```
void TextureHolder::Load (
    Textures::ID id,
    const std::string & filename )
```

Load the image from the file.

This function is used to load the image from the file.

**Parameters**

<i>id</i>	The image ID.
<i>filename</i>	The image filename.

**7.61.3.5 Load() [2/2]**

```
void TextureHolder::Load (
    Textures::ID id,
    const std::string & filename,
    int width,
    int height,
    bool cropCenter = false )
```

Load the texture from the file.

This function is used to load the texture from the file.

**Parameters**

<i>id</i>	The image ID.
<i>filename</i>	The image filename.
<i>width</i>	The image width.
<i>height</i>	The image height.
<i>cropCenter</i>	The flag that indicates whether the image should be cropped from the center.

**See also**[Texture::ID](#)**7.61.3.6 LoadFromImage() [1/2]**

```
void TextureHolder::LoadFromImage (
    Textures::ID id,
    Image & image )
```

Load the texture from the image.

This function is used to load the texture from the image.

**Parameters**

<i>id</i>	The image ID.
<i>image</i>	The image.

**See also**[Texture::ID](#)**7.61.3.7 LoadFromImage() [2/2]**

```
void TextureHolder::LoadFromImage (
    Textures::ID id,
    Image & image,
    int width,
    int height )
```

Load the texture from the image and crop it to the specified width and height.

This function is used to load the texture from the image and crop it to the specified width and height.

**Parameters**

<i>id</i>	The image ID.
<i>image</i>	The image.
<i>width</i>	The image width.
<i>height</i>	The image height.

**See also**[Texture::ID](#)

### 7.61.4 Member Data Documentation

#### 7.61.4.1 mTextureMap

```
std::map< Textures::ID, std::unique_ptr< Texture > > TextureHolder::mTextureMap [private]
```

The texture map.

The texture map that is used to store the textures. As the flyweight pattern, the textures are stored in the map and the textures are retrieved from the map.

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

- src/[TextureHolder.hpp](#)
- src/[TextureHolder.cpp](#)

## 7.62 GUIComponent::NavigationBar::TitleInfo Struct Reference

### Public Attributes

- [States::ID stateID](#)
- std::string [abbrTitle](#)
- std::string [titleName](#)

### 7.62.1 Member Data Documentation

#### 7.62.1.1 abbrTitle

```
std::string GUIComponent::NavigationBar::TitleInfo::abbrTitle
```

#### 7.62.1.2 stateID

```
States::ID GUIComponent::NavigationBar::TitleInfo::stateID
```

#### 7.62.1.3 titleName

```
std::string GUIComponent::NavigationBar::TitleInfo::titleName
```

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

- src/Components/Common/[NavigationBar.hpp](#)

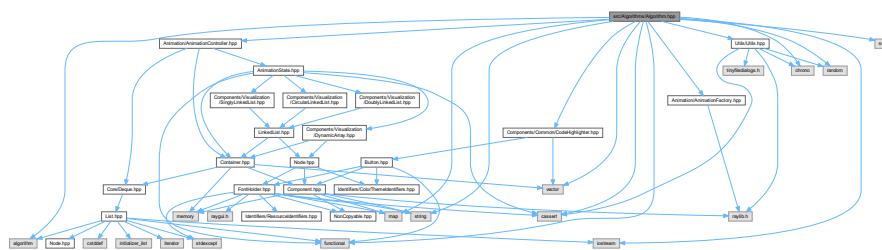


# Chapter 8

# File Documentation

## 8.1 src/Algorithms/Algorithm.hpp File Reference

```
#include <algorithm>
#include <cassert>
#include <chrono>
#include <functional>
#include <iostream>
#include <map>
#include <random>
#include <sstream>
#include <string>
#include <vector>
#include "Animation/AnimationController.hpp"
#include "Animation/AnimationFactory.hpp"
#include "Components/Common/CodeHighlighter.hpp"
#include "Utils/Utils.hpp"
Include dependency graph for Algorithm.hpp:
```



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



## Classes

- class [Algorithm::Algorithm< GUIAlgorithm, AnimationState >](#)

*Base class for all algorithms (which is used to generate step-by-step instructions for visualization)*

## Namespaces

- namespace [Algorithm](#)

## 8.2 Algorithm.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef CORE_ALGORITHMS_LINKEDLIST_ALGORITHM_HPP
00002 #define CORE_ALGORITHMS_LINKEDLIST_ALGORITHM_HPP
00003
00004 #include <algorithm>
00005 #include <cassert>
00006 #include <chrono>
00007 #include <functional>
00008 #include <iostream>
00009 #include <map>
00010 #include <random>
00011 #include <sstream>
00012 #include <string>
00013 #include <vector>
00014
00015 #include "Animation/AnimationController.hpp"
00016 #include "Animation/AnimationFactory.hpp"
00017 #include "Components/Common/CodeHighlighter.hpp"
00018 #include "Utils/Utils.hpp"
00019
00020 namespace Algorithm {
00021     template< typename GUIAlgorithm, typename AnimationState >
00022     class Algorithm {
00023     public:
00024         Algorithm(GUIComponent::CodeHighlighter::Ptr codeHighlighter,
00025                  typename Animation::AnimationController< AnimationState >::Ptr
00026                  animController,
00027                  FontHolder* fonts);
00028
00029         Algorithm();
00030
00031         ~Algorithm();
00032
00033     protected:
00034         GUIAlgorithm visualizer;
00035
00036         GUIComponent::CodeHighlighter::Ptr codeHighlighter;
00037
00038         typename Animation::AnimationController< AnimationState >::Ptr
00039         animController;
00040
00041     public:
00042         virtual void Empty();
00043
00044         virtual void Random();
00045
00046         virtual void RandomFixedSize(int N);
00047
00048         virtual void UserDefined(std::string input);
00049
00050         virtual void ReadFromExternalFile(std::string path);
00051
00052     protected:
00053         virtual void ApplyInput(std::vector< int > input,
00054                                std::size_t nMaxSize = 10);
00055
00056     protected:
00057         std::vector< int > EmptyGenerator();
00058
00059         std::vector< int > RandomGenerator();
00060
00061         std::vector< int > RandomFixedSizeGenerator(int nSize);
00062
00063         std::vector< int > UserDefinedGenerator(std::string input);
00064
00065     };
00066 }
```

```
00151         std::vector< int > ReadFromFileGenerator(std::string inputFile);
00152
00153     protected:
00154         virtual void GenerateRelayoutAnimation(Vector2 newPosition);
00155
00156         virtual AnimationState GenerateAnimation(float duration,
00157                                         int highlightLine,
00158                                         std::string actionDescription);
00159
00160         virtual void InitAction(std::vector< std::string > code);
00161     };
00162 } // namespace Algorithm
00163
00164 template< typename GUIAlgorithm, typename AnimationState >
00165 inline Algorithm::Algorithm< GUIAlgorithm, AnimationState >::Algorithm(
00166     GUIComponent::CodeHighlighter::Ptr codeHighlighter,
00167     typename Animation::AnimationController< AnimationState >::Ptr
00168     animController,
00169     FontHolder fonts)
00170 : codeHighlighter(codeHighlighter), animController(animController),
00171   visualizer(GUIAlgorithm(fonts)) {
00172     // visualizer.SetPosition(0, 150);
00173 }
00174
00175 template< typename GUIAlgorithm, typename AnimationState >
00176 Algorithm::Algorithm< GUIAlgorithm, AnimationState >::Algorithm() {}
00177
00178 template< typename GUIAlgorithm, typename AnimationState >
00179 Algorithm::Algorithm< GUIAlgorithm, AnimationState >::~Algorithm() {}
00180
00181 template< typename GUIAlgorithm, typename AnimationState >
00182 std::vector< int >
00183 Algorithm::Algorithm< GUIAlgorithm, AnimationState >::EmptyGenerator() {
00184     return std::vector< int >();
00185 }
00186
00187 template< typename GUIAlgorithm, typename AnimationState >
00188 std::vector< int >
00189 Algorithm::Algorithm< GUIAlgorithm, AnimationState >::RandomGenerator() {
00190     int nSize = Utils::Rand(1, 10);
00191     // params["nSize"] = (char)(nSize + '0');
00192     return RandomFixedSizeGenerator(nSize);
00193 }
00194
00195 template< typename GUIAlgorithm, typename AnimationState >
00196 Algorithm::Algorithm< GUIAlgorithm, AnimationState >::RandomFixedSizeGenerator(
00197     int nSize) {
00198     std::vector< int > answer(nSize);
00199     for (int& v : answer) v = Utils::Rand(1, 99);
00200     return answer;
00201 }
00202
00203 template< typename GUIAlgorithm, typename AnimationState >
00204 std::vector< int >
00205 Algorithm::Algorithm< GUIAlgorithm, AnimationState >::UserDefinedGenerator(
00206     std::string input) {
00207     bool canParse =
00208         (!input.empty() &&
00209          std::all_of(
00210              input.begin(), input.end(),
00211              [] (char c) { return ('0' <= c && c <= '9') || (c == ','); }) &&
00212          input.back() != ',');
00213     if (!canParse) return EmptyGenerator();
00214
00215     std::vector< int > answer;
00216
00217     std::istringstream f(input);
00218     std::string s;
00219     while (std::getline(f, s, ',')) {
00220         try {
00221             answer.push_back(atoi(s.c_str()));
00222         } catch (char* e) {
00223         }
00224     }
00225     return answer;
00226 }
00227
00228 template< typename GUIAlgorithm, typename AnimationState >
00229 std::vector< int >
00230 Algorithm::Algorithm< GUIAlgorithm, AnimationState >::ReadFromFileGenerator(
00231     std::string inputFile) {
00232     // will be implemented later
00233     return std::vector< int >();
00234 }
```

```
00267 template< typename GUIAlgorithm, typename AnimationState >
00268 inline void
00269 Algorithm::Algorithm< GUIAlgorithm, AnimationState >::GenerateRelayoutAnimation(
00270     Vector2 newPosition) {
00271     AnimationState animRelayout =
00272         GenerateAnimation(0.5, -1,
00273             "Re-layout the Linked List for visualization "
00274             "(not in the actual Linked "
00275             "List).\nThe whole process is still O(1).");
00276     animRelayout.SetAnimation(
00277         [this, newPosition](GUIAlgorithm srcDS, float playingAt, Vector2 base) {
00278             Vector2 newPos = srcDS.GetPosition();
00279             newPos.x += (newPosition.x - newPos.x) * playingAt;
00280             newPos.y += (newPosition.y - newPos.y) * playingAt;
00281             srcDS.SetPosition(newPos);
00282
00283             srcDS.Draw(base, playingAt);
00284
00285             return srcDS;
00286         });
00287     animController->AddAnimation(animRelayout);
00288 }
00289
00290 template< typename GUIAlgorithm, typename AnimationState >
00291 AnimationState
00292 Algorithm::Algorithm< GUIAlgorithm, AnimationState >::GenerateAnimation(
00293     float duration, int highlightLine, std::string actionDescription) {
00294     AnimationState animation;
00295     animation.SetDuration(duration);
00296     animation.SetHighlightLine(highlightLine);
00297     animation.SetSourceDataStructure(visualizer);
00298     animation.SetActionDescription(actionDescription);
00299     return animation;
00300 }
00301
00302 template< typename GUIAlgorithm, typename AnimationState >
00303 void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::InitAction(
00304     std::vector< std::string > code) {
00305     animController->Reset();
00306     // animController->Pause();
00307     animController->InteractionAllow();
00308     animController->Clear();
00309     codeHighlighter->AddCode(code);
00310     codeHighlighter->SetShowCode(true);
00311     codeHighlighter->SetShowAction(true);
00312 }
00313
00314 template< typename GUIAlgorithm, typename AnimationState >
00315 void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::Empty() {
00316     ApplyInput(EmptyGenerator());
00317 }
00318
00319 template< typename GUIAlgorithm, typename AnimationState >
00320 void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::Random() {
00321     ApplyInput(RandomGenerator());
00322 }
00323
00324 template< typename GUIAlgorithm, typename AnimationState >
00325 void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::RandomFixedSize(
00326     int N) {
00327     ApplyInput(RandomFixedSizeGenerator(N));
00328 }
00329
00330 template< typename GUIAlgorithm, typename AnimationState >
00331 void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::UserDefined(
00332     std::string input) {
00333     ApplyInput(UserDefinedGenerator(input));
00334 }
00335
00336 template< typename GUIAlgorithm, typename AnimationState >
00337 void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::ReadFromExternalFile(
00338     std::string path) {
00339     ApplyInput(ReadFromFileGenerator(path));
00340 }
00341
00342 template< typename GUIAlgorithm, typename AnimationState >
00343 void Algorithm::Algorithm< GUIAlgorithm, AnimationState >::ApplyInput(
00344     std::vector< int > input, std::size_t nMaxSize) {
00345     if (input.size() > nMaxSize) input.resize(nMaxSize);
00346     InitAction({});
00347
00348     codeHighlighter->SetShowCode(false);
00349     codeHighlighter->SetShowAction(false);
00350
00351     visualizer.Import(input);
00352
00353     AnimationState state;
```

```

00354     state.SetDuration(0.5);
00355     state.SetHighlightLine(-1);
00356     state.SetSourceDataStructure(visualizer);
00357     state.SetAnimation([this] (GUIAlgorithm srcDS, float playingAt,
00358                         Vector2 base) {
00359         auto& nodes = srcDS.GetList();
00360         for (GUVISUALIZER::Node& node : nodes) {
00361             node.SetRadius(AnimationFactory::ElasticOut(playingAt) * 20);
00362             node.SetValueFontSize(AnimationFactory::ElasticOut(playingAt) * 24);
00363             node.SetLabelFontSize(AnimationFactory::ElasticOut(playingAt) * 20);
00364         }
00365         srcDS.Draw(base, playingAt, true);
00366     });
00367 );
00368
00369 animController->AddAnimation(state);
00370 animController->Reset();
00371 animController->InteractionLock();
00372 }
00373
00374 #endif // CORE_ALGORITHMS_LINKEDLIST_ALGORITHM_HPP

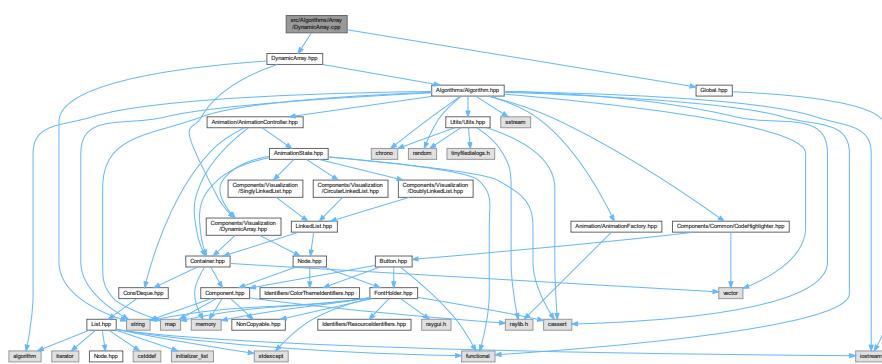
```

## 8.3 src/Algorithms/Array/DynamicArray.cpp File Reference

#include "DynamicArray.hpp"

#include "Global.hpp"

Include dependency graph for DynamicArray.cpp:



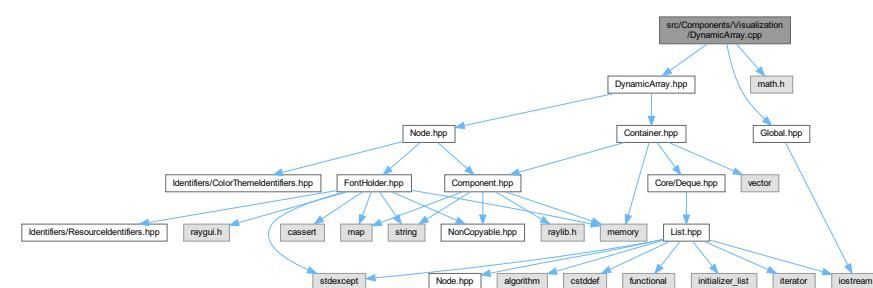
## 8.4 src/Components/Visualization/DynamicArray.cpp File Reference

#include "DynamicArray.hpp"

#include <math.h>

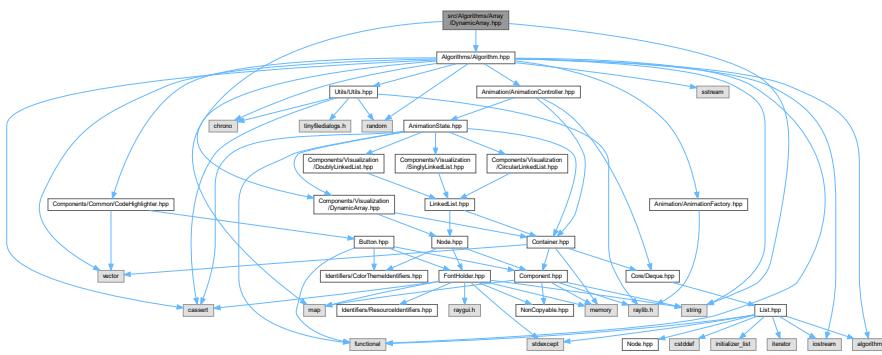
#include "Global.hpp"

Include dependency graph for DynamicArray.cpp:

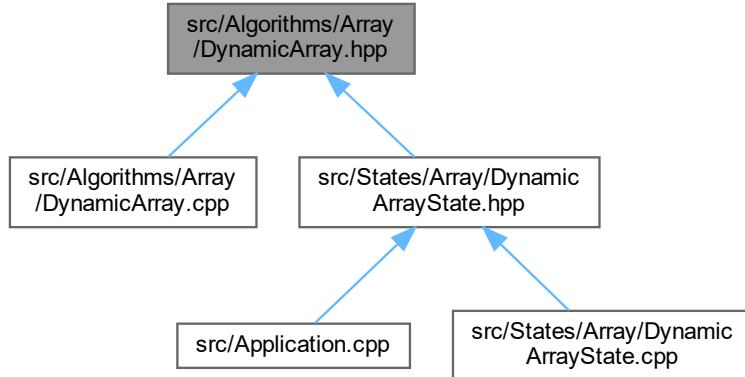


## 8.5 src/Algorithms/Array/DynamicArray.hpp File Reference

```
#include <string>
#include "Algorithms/Algorithm.hpp"
#include "Components/Visualization/DynamicArray.hpp"
Include dependency graph for DynamicArray.hpp:
```



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



### Classes

- class [Algorithm::DynamicArray](#)

*The algorithm class that is used to generate step-by-step instructions for the visualization of the dynamic array.*

### Namespaces

- namespace [Algorithm](#)

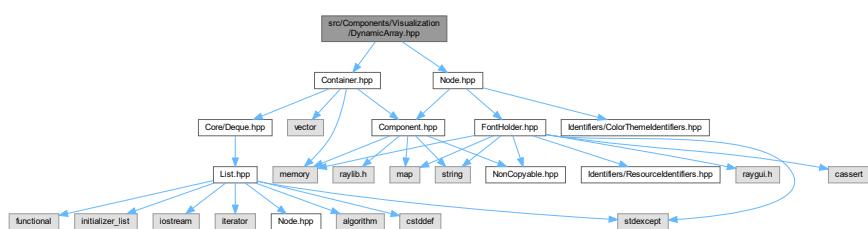
## 8.6 DynamicArray.hpp

[Go to the documentation of this file.](#)

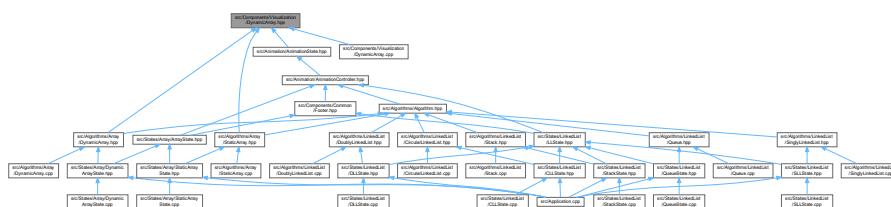
```
00001 #ifndef CORE_ALGORITHMS_ARRAY_DYNAMICARRAY_HPP
00002 #define CORE_ALGORITHMS_ARRAY_DYNAMICARRAY_HPP
00003
00004 #include <string>
00005
00006 // #include "BaseNode.hpp"
00007 #include "Algorithms/Algorithm.hpp"
00008 #include "Components/Visualization/DynamicArray.hpp"
00009
00010 namespace Algorithm {
00011     class DynamicArray
00012         : public Algorithm< GUIVisualizer::DynamicArray, DArrayAnimation > {
00013     public:
00014         static constexpr int maxN = 16;
00015
00016     public:
00017         DynamicArray();
00018
00019         DynamicArray(GUIComponent::CodeHighlighter::Ptr codeHighlighter,
00020                         DArrayAnimationController::Ptr animController,
00021                         FontHolder* fonts);
00022
00023         ~DynamicArray();
00024
00025         std::size_t size() const;
00026
00027     public:
00028         void Push(int index, int value);
00029
00030         void PushBack(int value);
00031
00032     public:
00033         void Remove(int index);
00034
00035         void PopBack();
00036
00037     public:
00038         void Update(int index, int value);
00039
00040     public:
00041         void Access(int index);
00042
00043     public:
00044         void Search(int value);
00045
00046     private:
00047         void ResetVisualizer();
00048     };
00049 }
00050 // namespace Algorithm
00051
00052 #endif // CORE_ALGORITHMS_ARRAY_DYNAMICARRAY_HPP
```

## 8.7 src/Components/Visualization/DynamicArray.hpp File Reference

```
#include "Container.hpp"
#include "Node.hpp"
Include dependency graph for DynamicArray.hpp:
```



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



## Classes

- class [GUIVisualizer::DynamicArray](#)

*The base class for the dynamic array visualization. This class provides the basic functionality for the dynamic array visualization.*

## Namespaces

- namespace [GUIVisualizer](#)

## 8.8 DynamicArray.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef COMPONENTS_VISUALIZATION_DYNAMICARRAY_HPP
00002 #define COMPONENTS_VISUALIZATION_DYNAMICARRAY_HPP
00003
00004 #include "Container.hpp"
00005 #include "Node.hpp"
00006
00007 namespace GUIVisualizer {
00012     class DynamicArray : public GUI::Container {
00013     public:
00017         DynamicArray();
00018
00023         DynamicArray(FontHolder* fonts);
00024
00025         bool isSelectable() const;
00026
00030         ~DynamicArray();
00031
00038         void Draw(Vector2 base = (Vector2){0, 0}, float t = 1.0f,
00039                   bool init = false);
00040
00041     public:
00046         std::size_t GetLength() const;
00047
00052         std::size_t GetCapacity() const;
00053
00059         Node& operator[](std::size_t index);
00060
00066         const Node& operator[](std::size_t index) const;
00067
00068     public:
00073         void SetShape(Node::Shape shape);
00074
00079         Node::Shape GetShape() const;
00080
00081     public:
00087         void Reserve(std::size_t size);
00088
00093         void Resize(std::size_t size);
00094
00098         void Clear();
00099
00100     public:
00105         std::vector< Node >& GetList();

```

```

00106     Node GenerateNode(int value);
00107
00114     void Import(std::vector< int > nodes);
00115
00123     void InsertNode(std::size_t index, Node node, bool rePosition = true);
00124
00131     void DeleteNode(std::size_t index, bool rePosition = true);
00132
00136     void Relayout();
00137
00138 private:
00139     Vector2 GetNodeDefaultPosition(std::size_t index);
00140
00141 public:
00142     std::size_t GetCapacityFromLength(std::size_t length);
00143
00144 public:
00145     static constexpr float mNodeDistance = 20;
00146
00147 private:
00148     FontHolder* fonts;
00149     std::vector< Node > list;
00150     Node::Shape mShape;
00151
00152 private:
00153     std::size_t capacity;
00154     std::size_t length;
00155 };
00156 }; // namespace GUIVisualizer
00157
00158 #endif // COMPONENTS_VISUALIZATION_DYNAMICARRAY_HPP

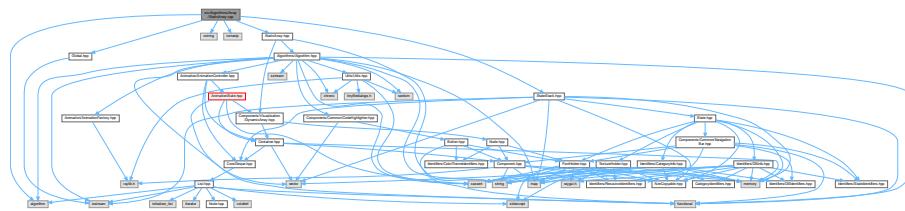
```

## 8.9 src/Algorithms/Array/StaticArray.cpp File Reference

```

#include "StaticArray.hpp"
#include <algorithm>
#include <cstring>
#include <iomanip>
#include "Global.hpp"
#include "StateStack.hpp"
Include dependency graph for StaticArray.cpp:

```



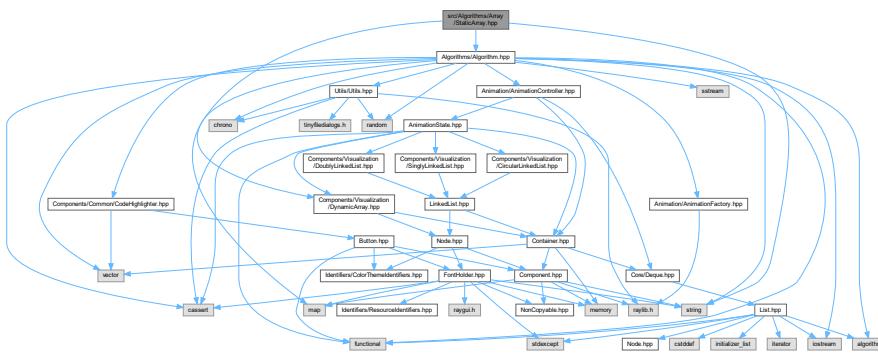
## 8.10 src/Algorithms/Array/StaticArray.hpp File Reference

```

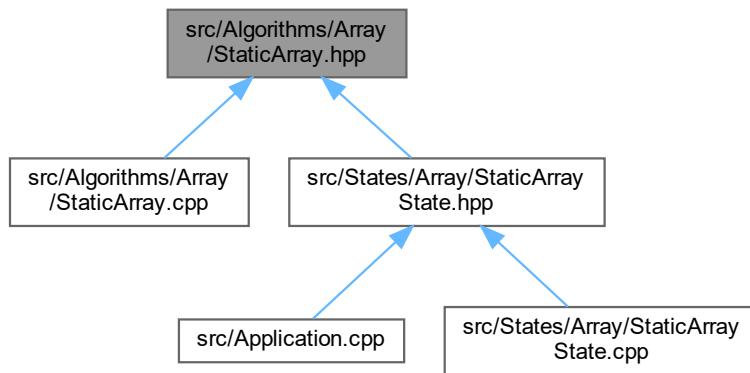
#include <string>
#include "Algorithms/Algorithm.hpp"
#include "Components/Visualization/DynamicArray.hpp"

```

Include dependency graph for StaticArray.hpp:



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



## Classes

- class [Algorithm::StaticArray](#)

*The algorithm class that is used to generate step-by-step instructions for the visualization of the static array.*

## Namespaces

- namespace [Algorithm](#)

## 8.11 StaticArray.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef CORE_ALGORITHMS_ARRAY_STATICARRAY_HPP
00002 #define CORE_ALGORITHMS_ARRAY_STATICARRAY_HPP
00003
00004 #include <string>
```

```

00005
00006 // #include "BaseNode.hpp"
00007 #include "Algorithms/Algorithm.hpp"
00008 #include "Components/Visualization/DynamicArray.hpp"
00009
00010 namespace Algorithm {
00011     class StaticArray
00012         : public Algorithm< GUIVisualizer::DynamicArray, DArrayAnimation > {
00013     public:
00014         static constexpr int maxN = 16;
00015
00016     public:
00017         StaticArray();
00018
00019         StaticArray(GUIComponent::CodeHighlighter::Ptr codeHighlighter,
00020                     DArrayAnimationController::Ptr animController,
00021                     FontHolder* fonts);
00022
00023         ~StaticArray();
00024
00025         std::size_t size() const;
00026
00027     public:
00028         void Insert(int index, int value);
00029
00030     public:
00031         void Delete(int index);
00032
00033     public:
00034         void Update(int index, int value);
00035
00036     public:
00037         void Access(int index);
00038
00039     public:
00040         void Search(int value);
00041
00042     private:
00043         void ApplyInput(std::vector< int > input, std::size_t nMaxSize);
00044
00045     private:
00046         void ResetVisualizer();
00047     };
00048 }; // namespace Algorithm
00049
00050 #endif // CORE_ALGORITHMS_ARRAY_STATICARRAY_HPP

```

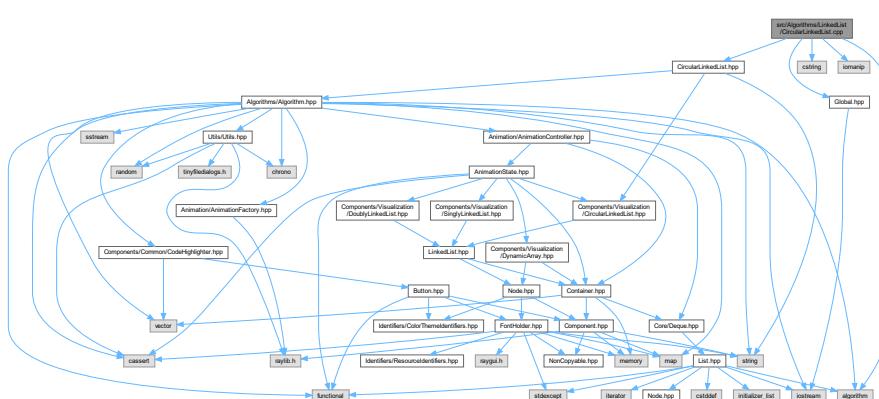
## 8.12 src/Algorithms/LinkedList/CircularLinkedList.cpp File Reference

```

#include "CircularLinkedList.hpp"
#include <algorithm>
#include <cstring>
#include <iomanip>
#include "Global.hpp"

```

Include dependency graph for CircularLinkedList.cpp:



## TypeDefs

- using `ArrowType = GUIVisualizer::LinkedList::ArrowType`

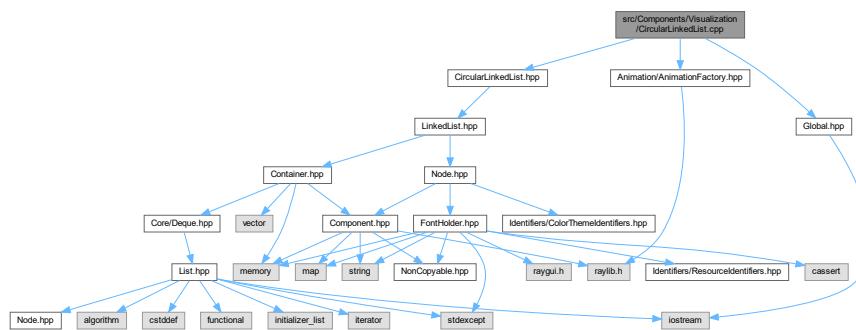
### 8.12.1 Typedef Documentation

### 8.12.1.1 ArrowType

```
using ArrowType = GUIVisualizer::LinkedList::ArrowType
```

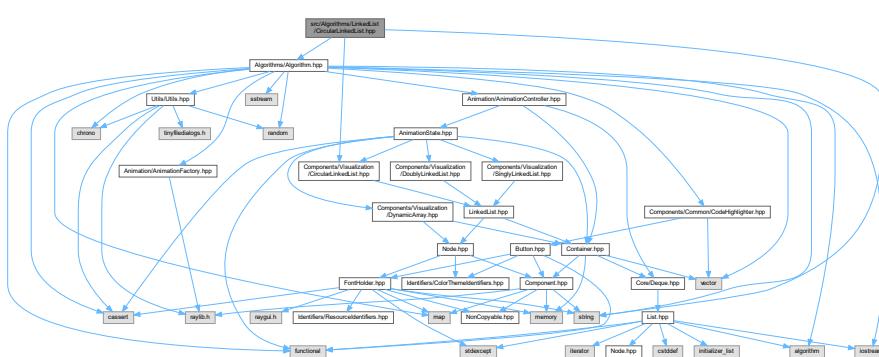
## 8.13 src/Components/Visualization/CircularLinkedList.cpp File Reference

```
#include "CircularLinkedList.hpp"
#include "Animation/AnimationFactory.hpp"
#include "Global.hpp"
Include dependency graph for CircularLinkedList.cpp:
```

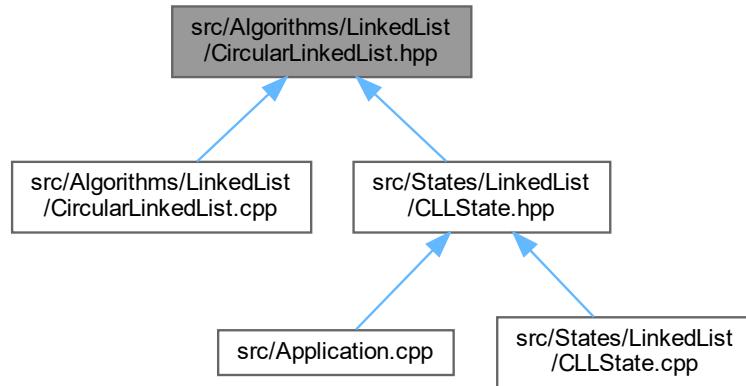


## 8.14 src/Algorithms/LinkedList/CircularLinkedList.hpp File Reference

```
#include <string>
#include "Algorithms/Algorithm.hpp"
#include "Components/Visualization/CircularLinkedList.hpp"
Include dependency graph for CircularLinkedList.hpp:
```



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



## Classes

- class [Algorithm::CircularLinkedList](#)

*The algorithm class that is used to generate step-by-step instructions for the visualization of the circular linked list.*

## Namespaces

- namespace [Algorithm](#)

## 8.15 CircularLinkedList.hpp

[Go to the documentation of this file.](#)

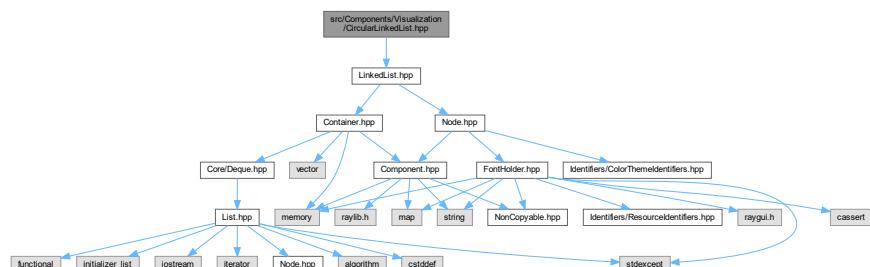
```

00001 #ifndef CORE_DATASTRUCTURES_LINKEDLIST_CIRCULARLINKEDLIST_HPP
00002 #define CORE_DATASTRUCTURES_LINKEDLIST_CIRCULARLINKEDLIST_HPP
00003
00004 #include <string>
00005
00006 // #include "BaseNode.hpp"
00007 #include "Algorithms/Algorithm.hpp"
00008 #include "Components/Visualization/CircularLinkedList.hpp"
00009
00010 namespace Algorithm {
00011     class CircularLinkedList
00012         : public Algorithm< GUIVisualizer::CircularLinkedList, CLLAnimation > {
00013     public:
00014         static constexpr int maxN = 16;
00015
00016         CircularLinkedList();
00017
00018         CircularLinkedList(GUIComponent::CodeHighlighter::Ptr codeHighlighter,
00019                            CLLAnimationController::Ptr animController,
00020                            FontHolder* fonts);
00021
00022         ~CircularLinkedList();
00023
00024         std::size_t size() const;
00025
00026     public:
00027         void InsertHead(int value);
  
```

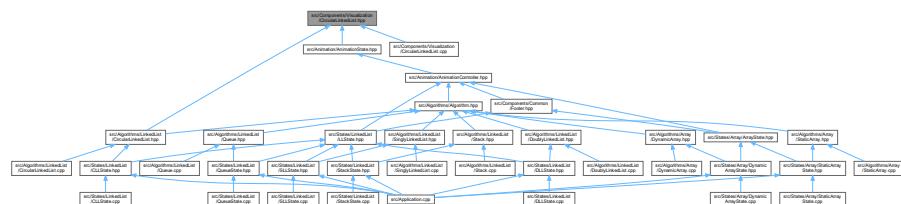
```
00062     void InsertAfterTail(int value);
00068     void InsertMiddle(int index, int value);
00081 public:
00086     void DeleteHead();
00087
00091     void DeleteTail();
00092
00100     void DeleteMiddle(int index);
00101
00102 public:
00107     void Update(int index, int value);
00108
00109 public:
00114     void Search(int value);
00115
00116 private:
00127     std::function< GUIVisualizer::CircularLinkedList(
00128         GUIVisualizer::CircularLinkedList, float, Vector2) >
00129     HighlightArrowFromCur(int index, bool drawVisualizer = true,
00130                           bool reverse = false);
00131
00141     std::function< GUIVisualizer::CircularLinkedList(
00142         GUIVisualizer::CircularLinkedList, float, Vector2) >
00143     HighlightCircularArrow(bool drawVisualizer = true,
00144                            bool reverse = false);
00145
00146 private:
00153     void ResetVisualizer();
00154 };
00155 }; // namespace Algorithm
00156
00157 #endif // CORE_DATASTRUCTURES_LINKEDLIST_CIRCULARLINKEDLIST_HPP
```

## 8.16 src/Components/Visualization/CircularLinkedList.hpp File Reference

```
#include "CircularLinkedList.hpp"
```



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



## Classes

- class [GUILabelizer::CircularLinkedList](#)

*The circular linked list visualization.*

## Namespaces

- namespace [GUILabelizer](#)

## 8.17 CircularLinkedList.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef COMPONENTS_VISUALIZATION_CIRCULARLINKEDLIST_HPP
00002 #define COMPONENTS_VISUALIZATION_CIRCULARLINKEDLIST_HPP
00003
00004 #include "LinkedList.hpp"
00005
00006 namespace GUILabelizer {
00012     class CircularLinkedList : public GUILabelizer::LinkedList {
00013     private:
00014         std::vector< ArrowType > arrowState;
00015
00016     private:
00017         ArrowType circularArrowState;
00018         std::pair< std::size_t, std::size_t > mCircularEnds;
00019
00020     public:
00024         CircularLinkedList();
00025
00029         CircularLinkedList(FontHolder* fonts);
00030
00034         ~CircularLinkedList();
00035
00036         bool isSelectable() const;
00037
00047         void Draw(Vector2 base = (Vector2){0, 0}, float t = 1.0f,
00048                 bool init = false);
00049
00050     public:
00058         void Import(std::vector< int > nodes);
00059
00068         void InsertNode(std::size_t index, Node node, bool rePosition = true);
00069
00077         void DeleteNode(std::size_t index, bool rePosition = true);
00078
00079     public:
00080         void SetCircularArrowType(ArrowType type);
00081         ArrowType GetCircularArrowType(std::size_t index);
00082         void SetCircularEnds(std::size_t from, std::size_t to);
00083         std::pair< std::size_t, std::size_t > GetCircularEnds();
00084
00085         void SetArrowType(std::size_t index, ArrowType type);
00086         ArrowType GetArrowType(std::size_t index);
00087         void ResetArrow();
00088
00089     private:
00090         void DrawArrow(Vector2 base, float t);
00091     };
00092 }; // namespace GUILabelizer
00093
00094 #endif // COMPONENTS_VISUALIZATION_CIRCULARLINKEDLIST_HPP

```

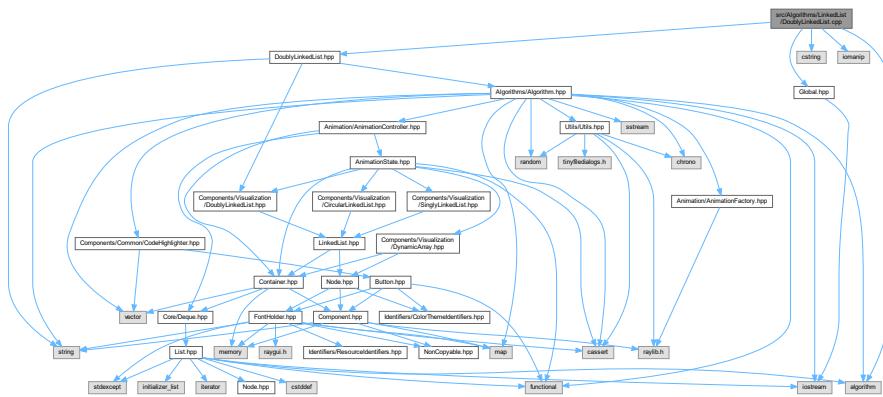
## 8.18 src/Algorithms/LinkedList/DoublyLinkedList.cpp File Reference

```

#include "DoublyLinkedList.hpp"
#include <algorithm>
#include <cstring>
#include <iomanip>

```

```
#include "Global.hpp"
Include dependency graph for DoublyLinkedList.cpp:
```



## TypeDefs

- using `ArrowType = GUIVisualizer::LinkedList::ArrowType`

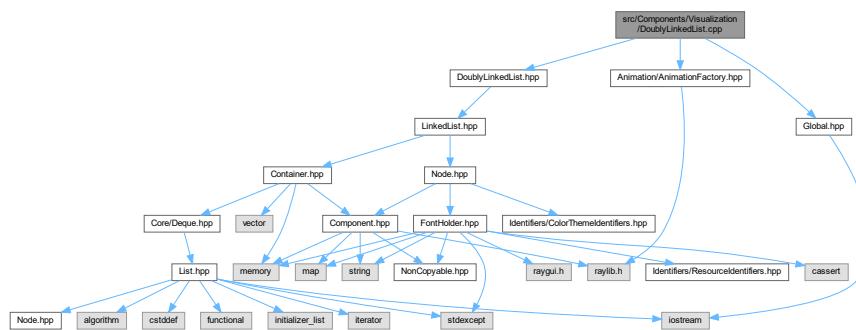
### 8.18.1 Typedef Documentation

### 8.18.1.1 ArrowType

```
using ArrowType = GUIVisualizer::LinkedList::ArrowType
```

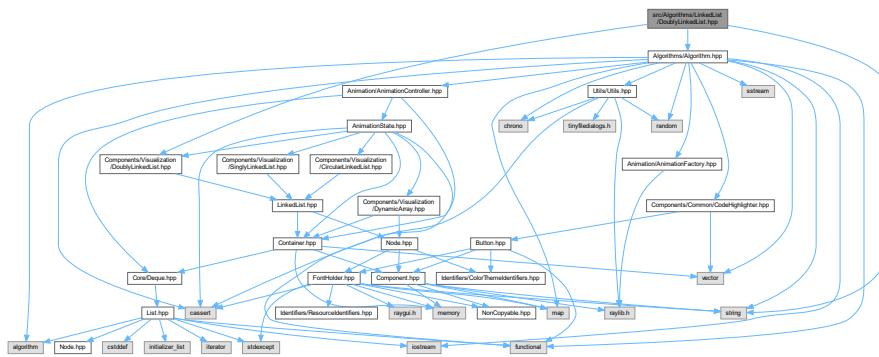
## 8.19 src/Components/Visualization/DoublyLinkedList.cpp File Reference

```
#include "DoublyLinkedList.hpp"
#include "Animation/AnimationFactory.hpp"
#include "Global.hpp"
Include dependency graph for DoublyLinkedList.cpp:
```

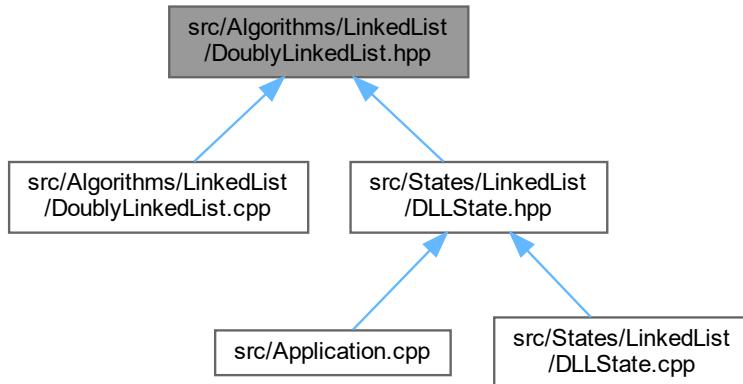


## 8.20 src/Algorithms/LinkedList/DoublyLinkedList.hpp File Reference

```
#include <string>
#include "Algorithms/Algorithm.hpp"
#include "Components/Visualization/DoublyLinkedList.hpp"
Include dependency graph for DoublyLinkedList.hpp:
```



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



## Classes

- class Algorithm::DoublyLinkedList

*The algorithm class that is used to generate step-by-step instructions for the visualization of the doubly linked list.*

## Namespaces

- namespace [Algorithm](#)

## 8.21 DoublyLinkedList.hpp

[Go to the documentation of this file.](#)

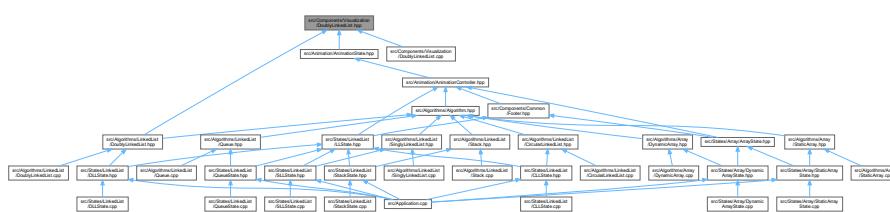
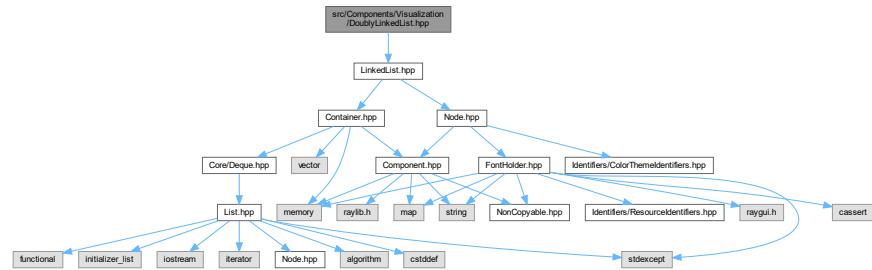
```

00001 #ifndef CORE_DATASTRUCTURES_LINKEDLIST_DOUBLYLINKEDLIST_HPP
00002 #define CORE_DATASTRUCTURES_LINKEDLIST_DOUBLYLINKEDLIST_HPP
00003
00004 #include <string>
00005
00006 // #include "BaseNode.hpp"
00007 #include "Algorithms/Algorithm.hpp"
00008 #include "Components/Visualization/DoublyLinkedList.hpp"
00009
00010 namespace Algorithm {
00011     class DoublyLinkedList
00012         : public Algorithm< GUIVisualizer::DoublyLinkedList, DLLAnimation > {
00013     public:
00014         static constexpr int maxN = 16;
00015
00016     public:
00017         DoublyLinkedList();
00018
00019         DoublyLinkedList(GUIComponent::CodeHighlighter::Ptr codeHighlighter,
00020                         DLLAnimationController::Ptr animController,
00021                         FontHolder* fonts);
00022
00023         ~DoublyLinkedList();
00024
00025         std::size_t size() const;
00026
00027     public:
00028         void InsertHead(int value);
00029
00030         void InsertAfterTail(int value);
00031
00032         void InsertMiddle(int index, int value);
00033
00034     public:
00035         void DeleteHead();
00036
00037         void DeleteTail();
00038
00039         void DeleteMiddle(int index);
00040
00041     public:
00042         void Update(int index, int value);
00043
00044     public:
00045         void Search(int value);
00046
00047     private:
00048         std::function< GUIVisualizer::DoublyLinkedList(
00049             GUIVisualizer::DoublyLinkedList, float, Vector2) >
00050             HighlightArrowNext(int index, bool drawVisualizer = true,
00051                               bool reverse = false);
00052
00053         std::function< GUIVisualizer::DoublyLinkedList(
00054             GUIVisualizer::DoublyLinkedList, float, Vector2) >
00055             HighlightArrowPrev(int index, bool drawVisualizer = true,
00056                               bool reverse = false);
00057
00058         std::function< GUIVisualizer::DoublyLinkedList(
00059             GUIVisualizer::DoublyLinkedList, float, Vector2) >
00060             HighlightArrowBoth(int index, bool drawVisualizer = true,
00061                               bool reverse = false);
00062
00063     private:
00064         void ResetVisualizer();
00065     };
00066 }; // namespace Algorithm
00067
00068#endif // CORE_DATASTRUCTURES_LINKEDLIST_DOUBLYLINKEDLIST_HPP

```

## 8.22 src/Components/Visualization/DoublyLinkedList.hpp File Reference

```
#include "LinkedList.hpp"
Include dependency graph for DoublyLinkedList.hpp:
```



## Classes

- class [GUIVisualizer::DoublyLinkedList](#)

*The doubly linked list visualization.*

## Namespaces

- namespace [GUIVisualizer](#)

## 8.23 DoublyLinkedList.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef COMPONENTS_VISUALIZATION_DOUBLYLINKEDLIST_HPP
00002 #define COMPONENTS_VISUALIZATION_DOUBLYLINKEDLIST_HPP
00003
00004 #include "LinkedList.hpp"
00005
00006 namespace GUIVisualizer {
00012     class DoublyLinkedList : public GUIVisualizer::LinkedList {
00013     private:
00014         std::vector< ArrowType > arrowStateNext;
00015         std::vector< ArrowType > arrowStatePrev;
00016
00017     public:
00021         DoublyLinkedList();
00022
00026         DoublyLinkedList(FontHolder* fonts);
  
```

```

00027
00031     ~DoublyLinkedList();
00032
00033     bool isSelectable() const;
00034
00044     void Draw(Vector2 base = (Vector2){0, 0}, float t = 1.0f,
00045         bool init = false);
00046
00047 public:
00048     void Import(std::vector< int > nodes);
00049
00056     void InsertNode(std::size_t index, Node node, bool rePosition = true);
00063     void DeleteNode(std::size_t index, bool rePosition = true);
00075
00076 public:
00077     void SetArrowTypeNext(std::size_t index, ArrowType type);
00078     void SetArrowTypePrev(std::size_t index, ArrowType type);
00079     ArrowType GetArrowTypeNext(std::size_t index);
00080     ArrowType GetArrowTypePrev(std::size_t index);
00081     void ResetArrow();
00082
00083 private:
00084     void DrawArrow(Vector2 base, float t);
00085 };
00086 }; // namespace GUIVisualizer
00087
00088 #endif // COMPONENTS_VISUALIZATION_DOUBLYLINKEDLIST_HPP

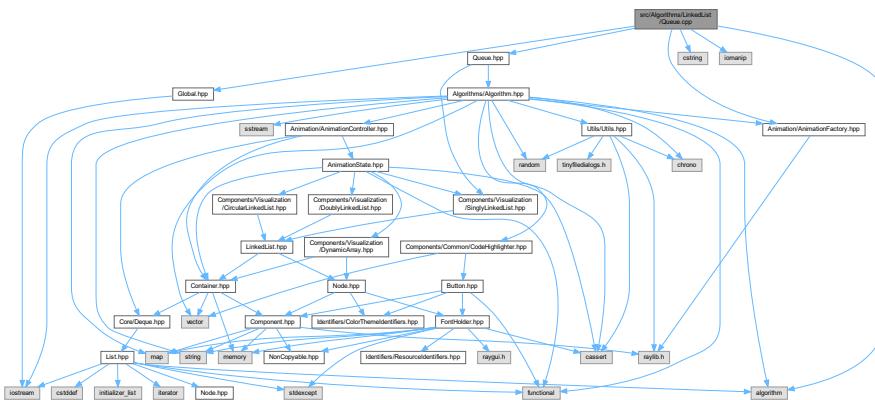
```

## 8.24 src/Algorithms/LinkedList/Queue.cpp File Reference

```

#include "Queue.hpp"
#include <algorithm>
#include <cstring>
#include <iomanip>
#include "Animation/AnimationFactory.hpp"
#include "Global.hpp"
Include dependency graph for Queue.cpp:

```



### Typedefs

- using **ArrowType** = **GUIVisualizer::SinglyLinkedList::ArrowType**

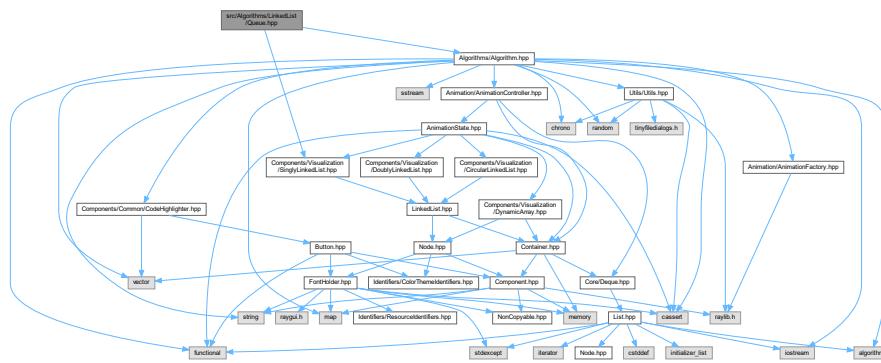
#### 8.24.1 Typedef Documentation

### 8.24.1.1 ArrowType

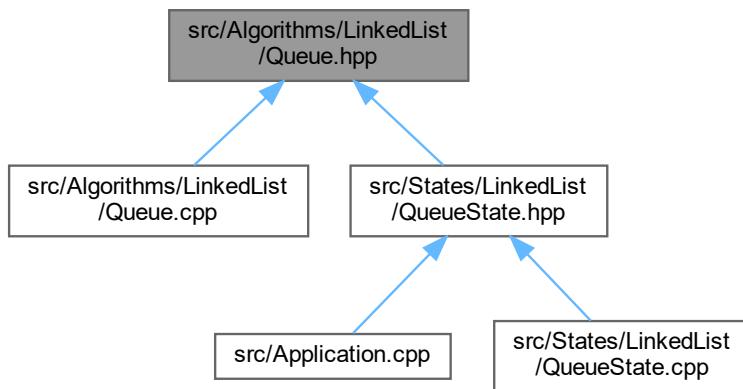
```
using ArrowType = GUIVisualizer::SinglyLinkedList::ArrowType
```

## 8.25 src/Algorithms/LinkedList/Queue.hpp File Reference

```
#include "Algorithms/Algorithm.hpp"
#include "Components/Visualization/SinglyLinkedList.hpp"
Include dependency graph for Queue.hpp:
```



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



## Classes

- class [Algorithm::Queue](#)

*The algorithm class that is used to generate step-by-step instructions for the visualization of the queue.*

## Namespaces

- namespace Algorithm

## 8.26 Queue.hpp

[Go to the documentation of this file.](#)

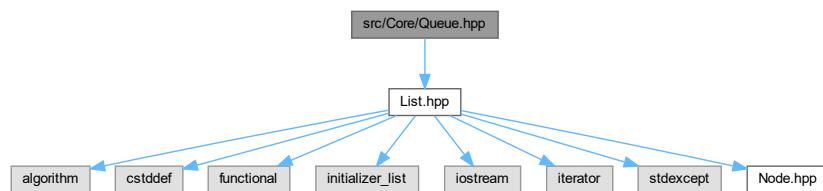
```

00001 #ifndef CORE_ALGORITHMS_LINKEDLIST_QUEUE_HPP
00002 #define CORE_ALGORITHMS_LINKEDLIST_QUEUE_HPP
00003
00004 #include "Algorithms/Algorithm.hpp"
00005 #include "Components/Visualization/SinglyLinkedList.hpp"
00006
00007 namespace Algorithm {
00012     class Queue
00013         : public Algorithm< GUIVisualizer::SinglyLinkedList, SLLAnimation > {
00014     public:
00018         static constexpr int maxN = 16;
00019
00020     public:
00028         Queue(GUIComponent::CodeHighlighter::Ptr _codeHighlighter,
00029                 SLLAnimationController::Ptr animController, FontHolder* fonts);
00030
00036     Queue();
00037
00042     ~Queue();
00043
00048     std::size_t size() const;
00049
00050     public:
00055     void EnqueueEmpty(int value);
00056
00062     void Enqueue(int value);
00063
00064     public:
00069     void Dequeue();
00070
00071     public:
00077     void PeekFront();
00078
00084     void PeekBack();
00085
00086     private:
00093     std::function< GUIVisualizer::SinglyLinkedList(
00094             GUIVisualizer::SinglyLinkedList, float, Vector2) >
00095     HighlightArrowFromCur(int index, bool drawVisualizer = true,
00096                           bool reverse = false);
00097 };
00098 }; // namespace Algorithm
00099
00100 #endif // CORE_ALGORITHMS_LINKEDLIST_QUEUE_HPP

```

## 8.27 src/Core/Queue.hpp File Reference

```
#include "List.hpp"
Include dependency graph for Queue.hpp:
```



## Classes

- class `Core::Queue< T >`

*The queue container.*

## Namespaces

- namespace `Core`

## 8.28 Queue.hpp

[Go to the documentation of this file.](#)

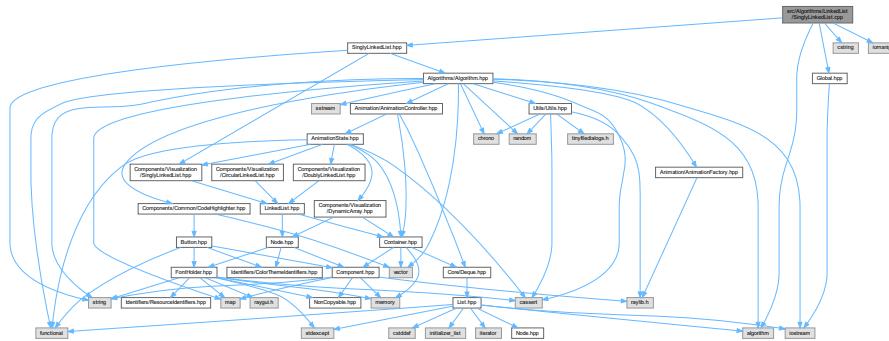
```
00001 #ifndef CORE_QUEUE_HPP
00002 #define CORE_QUEUE_HPP
00003
00004 #include "List.hpp"
00005
00006 namespace Core {
00007     template< typename T >
00008     class Queue : public List< T > {
00009     private:
00010         using List = Core::List< T >;
00011
00012     public:
00013         using List::List;
00014         using List::operator=;
00015         using List::swap;
00016
00017     public:
00018         using List::empty;
00019         using List::size;
00020
00021     public:
00022         T& front() { return List::front(); };
00023
00024     const T& front() const { return List::front(); };
00025
00026     T& back() { return List::back(); };
00027
00028     const T& back() const { return List::back(); };
00029
00030     public:
00031         void push(const T& value) { List::push_back(value); };
00032
00033         void push(T& value) { List::push_back(std::move(value)); };
00034
00035     public:
00036         void pop() { List::pop_front(); };
00037     };
00038 }; // namespace Core
00039
00040 #endif // CORE_QUEUE_HPP
```

## 8.29 src/Algorithms/LinkedList/SinglyLinkedList.cpp File Reference

```
#include "SinglyLinkedList.hpp"
#include <algorithm>
#include <cstring>
#include <iomanip>
```

```
#include "Global.hpp"
```

Include dependency graph for SinglyLinkedList.cpp:



## TypeDefs

- using `ArrowType = GUIVisualizer::LinkedList::ArrowType`

### 8.29.1 Typedef Documentation

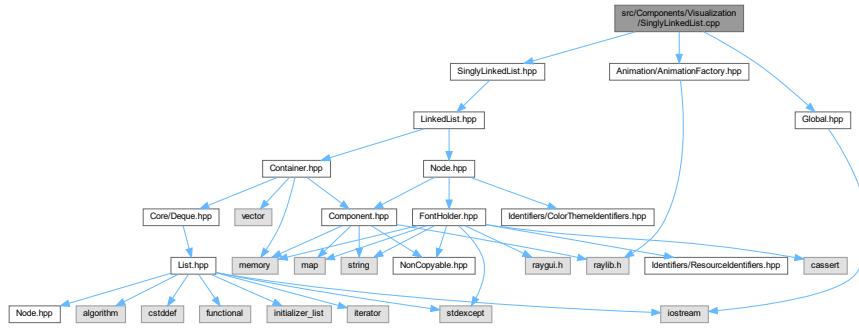
### 8.29.1.1 ArrowType

```
using ArrowType = GUIVisualizer::LinkedList::ArrowType
```

## 8.30 src/Components/Visualization/SinglyLinkedList.cpp File Reference

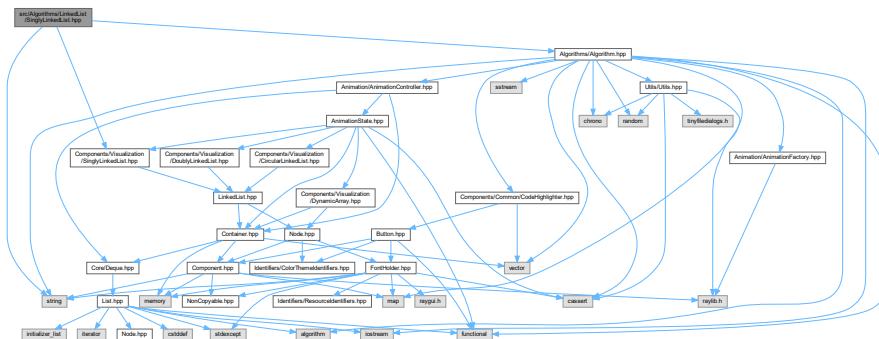
```
#include "SinglyLinkedList.hpp"
#include "Animation/AnimationFactory.hpp"
#include "Global.hpp"
Include dependency graph for SinglyLinkedList.cpp:
```

Include dependency graph for SinglyLinkedList.cpp:

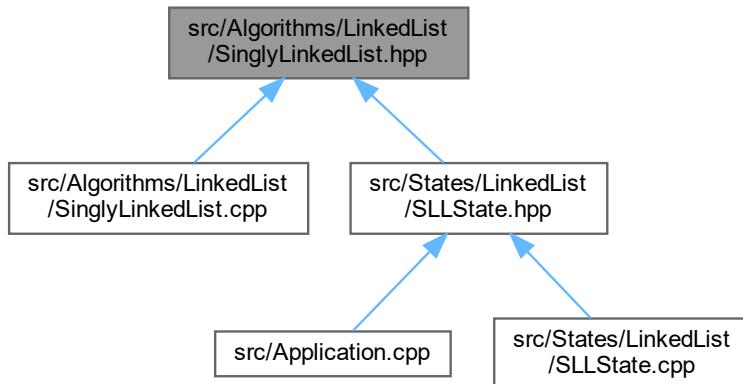


## 8.31 src/Algorithms/LinkedList/SinglyLinkedList.hpp File Reference

```
#include <string>
#include "Algorithms/Algorithm.hpp"
#include "Components/Visualization/SinglyLinkedList.hpp"
Include dependency graph for SinglyLinkedList.hpp:
```



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



## Classes

- class [Algorithm::SinglyLinkedList](#)

*The algorithm class that is used to generate step-by-step instructions for the visualization of the singly linked list.*

## Namespaces

- namespace [Algorithm](#)

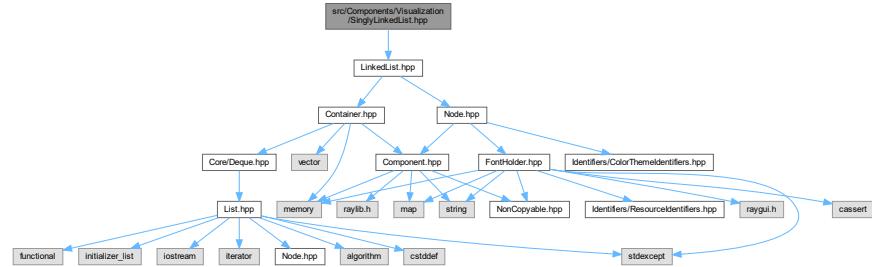
## 8.32 SinglyLinkedList.hpp

[Go to the documentation of this file.](#)

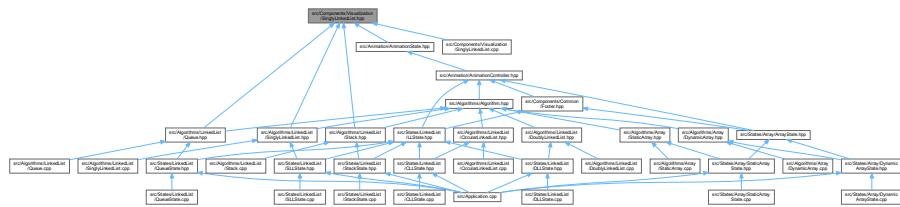
```
00001 #ifndef CORE_DATASTRUCTURES_LINKEDLIST_SINGLYLINKEDLIST_HPP
00002 #define CORE_DATASTRUCTURES_LINKEDLIST_SINGLYLINKEDLIST_HPP
00003
00004 #include <string>
00005
00006 // #include "BaseNode.hpp"
00007 #include "Algorithms/Algorithm.hpp"
00008 #include "Components/Visualization/SinglyLinkedList.hpp"
00009
00010 namespace Algorithm {
00011     class SinglyLinkedList
00012         : public Algorithm< GUIVisualizer::SinglyLinkedList, SLLAnimation > {
00013     public:
00014         static constexpr int maxN = 16;
00015
00016     public:
00017         SinglyLinkedList();
00018
00019         SinglyLinkedList(GUIComponent::CodeHighlighter::Ptr codeHighlighter,
00020                         SLAnimationController::Ptr animController,
00021                         FontHolder* fonts);
00022
00023         ~SinglyLinkedList();
00024
00025         std::size_t size() const;
00026
00027     public:
00028         void InsertHead(int value);
00029
00030         void InsertAfterTail(int value);
00031
00032         void InsertMiddle(int index, int value);
00033
00034     public:
00035         void DeleteHead();
00036
00037         void DeleteTail();
00038
00039         void DeleteMiddle(int index);
00040
00041     public:
00042         void Update(int index, int value);
00043
00044     public:
00045         void Search(int value);
00046
00047     private:
00048         std::function< GUIVisualizer::SinglyLinkedList(
00049             GUIVisualizer::SinglyLinkedList, float, Vector2) >
00050             HighlightArrowFromCur(int index, bool drawVisualizer = true,
00051             bool reverse = false);
00052
00053     private:
00054         void ResetVisualizer();
00055     };
00056 }; // namespace Algorithm
00057
00058#endif // CORE_DATASTRUCTURES_LINKEDLIST_SINGLYLINKEDLIST_HPP
```

## 8.33 src/Components/Visualization/SinglyLinkedList.hpp File Reference

```
#include "LinkedList.hpp"
Include dependency graph for SinglyLinkedList.hpp:
```



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



## Classes

- class [GUIVisualizer::SinglyLinkedList](#)

*The singly linked list visualization.*

## Namespaces

- namespace [GUIVisualizer](#)

## 8.34 SinglyLinkedList.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef COMPONENTS_VISUALIZATION_SINGLYLINKEDLIST_HPP
00002 #define COMPONENTS_VISUALIZATION_SINGLYLINKEDLIST_HPP
00003
00004 #include "LinkedList.hpp"
00005
00006 namespace GUIVisualizer {
00012     class SinglyLinkedList : public GUIVisualizer::LinkedList {
00013     private:
00014         std::vector< ArrowType > arrowState;
00015
00016     public:
00020         SinglyLinkedList();
00021
00025         SinglyLinkedList(FontHolder* fonts);
00026 }
```

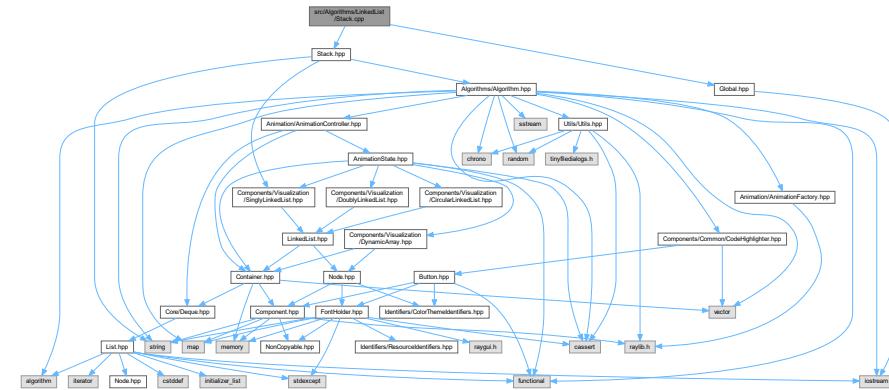
```

00030     ~SinglyLinkedList();
00031
00032     bool IsSelectable() const;
00033
00034     void Draw(Vector2 base = (Vector2){0, 0}, float t = 1.0f,
00035             bool init = false);
00036
00037 public:
00038     void Import(std::vector< int > nodes);
00039
00040     void InsertNode(std::size_t index, Node node, bool rePosition = true);
00041
00042     void DeleteNode(std::size_t index, bool rePosition = true);
00043
00044 public:
00045     void SetArrowType(std::size_t index, ArrowType type);
00046     ArrowType GetArrowType(std::size_t index);
00047     void ResetArrow();
00048
00049 private:
00050     void DrawArrow(Vector2 base, float t);
00051 };
00052 // namespace GUIVisualizer
00053
00054 #endif // COMPONENTS_VISUALIZATION_SINGLYLINKEDLIST_HPP

```

## 8.35 src/Algorithms/LinkedList/Stack.cpp File Reference

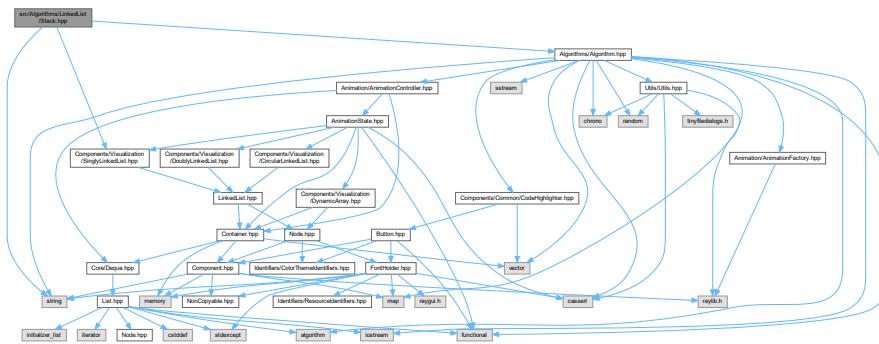
```
#include "Stack.hpp"
#include "Global.hpp"
Include dependency graph for Stack.cpp:
```



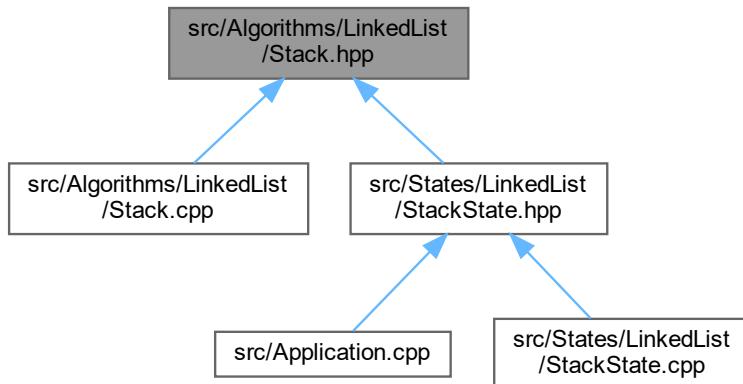
## 8.36 src/Algorithms/LinkedList/Stack.hpp File Reference

```
#include <string>
#include "Algorithms/Algorithm.hpp"
#include "Components/Visualization/SinglyLinkedList.hpp"
```

Include dependency graph for Stack.hpp:



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



## Classes

- class [Algorithm::Stack](#)

*The algorithm class that is used to generate step-by-step instructions for the visualization of the stack.*

## Namespaces

- namespace [Algorithm](#)

## Typedefs

- using [ArrowType](#) = [GUIVisualizer::SinglyLinkedList::ArrowType](#)
- using [Orientation](#) = [GUIVisualizer::LinkedList::Orientation](#)

## 8.36.1 Typedef Documentation

### 8.36.1.1 ArrowType

```
using ArrowType = GUIVisualizer::SinglyLinkedList::ArrowType
```

### 8.36.1.2 Orientation

```
using Orientation = GUIVisualizer::LinkedList::Orientation
```

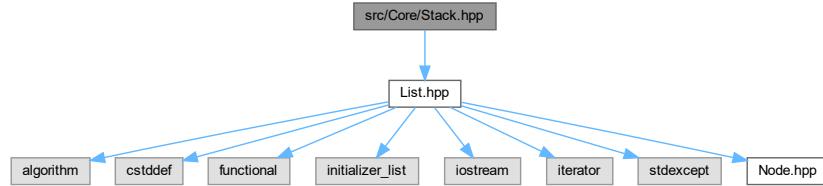
## 8.37 Stack.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef CORE_ALGORITHMS_LINKEDLIST_STACK_HPP
00002 #define CORE_ALGORITHMS_LINKEDLIST_STACK_HPP
00003
00004 #include <string>
00005
00006 // #include "BaseNode.hpp"
00007 #include "Algorithms/Algorithm.hpp"
00008 #include "Components/Visualization/SinglyLinkedList.hpp"
00009
00010 using ArrowType = GUIVisualizer::SinglyLinkedList::ArrowType;
00011 using Orientation = GUIVisualizer::LinkedList::Orientation;
00012
00013 namespace Algorithm {
00014     class Stack
00015         : public Algorithm< GUIVisualizer::SinglyLinkedList, SLLAnimation > {
00016     public:
00017         static constexpr int maxN = 10;
00018
00019         static constexpr Orientation mStackOrientation = Orientation::Vertical;
00020
00021     public:
00022         Stack(GUIComponent::CodeHighlighter::Ptr codeHighlighter,
00023               SLLAnimationController::Ptr animController, FontHolder* fonts);
00024
00025         Stack();
00026
00027         ~Stack();
00028
00029         std::size_t size() const;
00030
00031     public:
00032         void Push(int value);
00033
00034     public:
00035         void Pop();
00036
00037     public:
00038         void Peek();
00039
00040     private:
00041         std::function< GUIVisualizer::SinglyLinkedList(
00042             GUIVisualizer::SinglyLinkedList, float, Vector2) >
00043             HighlightArrowFromCur(int index, bool drawVisualizer = true,
00044                                   bool reverse = false);
00045     };
00046 } // namespace Algorithm
00047
00048#endif // CORE_ALGORITHMS_LINKEDLIST_STACK_HPP
```

## 8.38 src/Core/Stack.hpp File Reference

```
#include "List.hpp"
Include dependency graph for Stack.hpp:
```



### Classes

- class [Core::Stack< T >](#)  
*The stack container.*

### Namespaces

- namespace [Core](#)

## 8.39 Stack.hpp

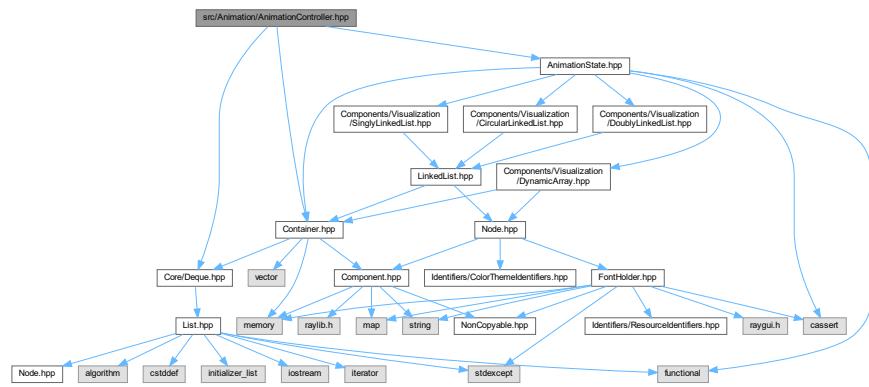
[Go to the documentation of this file.](#)

```

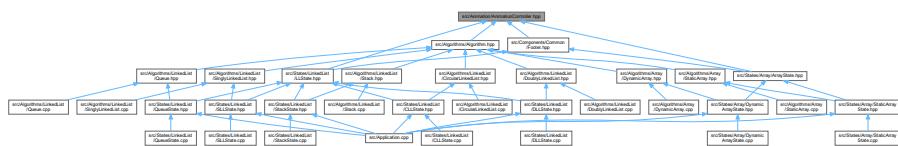
00001 #ifndef CORE_STACK_HPP
00002 #define CORE_STACK_HPP
00003
00004 #include "List.hpp"
00005
00006 namespace Core {
00011     template< typename T >
00012     class Stack : public List< T > {
00013     private:
00014         using List = Core::List< T >;
00015
00016     public:
00017         using List::List;
00018         using List::operator=;
00019         using List::swap;
00020
00021     public:
00022         using List::empty;
00023         using List::size;
00024
00025     public:
00030         T& top() { return List::front(); };
00031
00036         const T& top() const { return List::front(); };
00037
00038     public:
00043         void push(const T& value) { List::push_front(value); };
00044
00049         void push(T& value) { List::push_front(std::move(value)); };
00050
00051     public:
00055         void pop() { List::pop_front(); };
00056     };
00057 }; // namespace Core
00058
00059 #endif // CORE_STACK_HPP
  
```

## 8.40 src/Animation/AnimationController.hpp File Reference

```
#include "AnimationState.hpp"
#include "Container.hpp"
#include "Core/Deque.hpp"
Include dependency graph for AnimationController.hpp:
```



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



## Classes

- class Animation::AnimationController< T >  
*The animation controller class.*

## Namespaces

- namespace Animation

## TypeDefs

- `typedef Animation::AnimationController< SLLAnimation > SLLAnimationController`
  - `typedef Animation::AnimationController< DLLAnimation > DLLAnimationController`
  - `typedef Animation::AnimationController< CLLAnimation > CLLAnimationController`
  - `typedef Animation::AnimationController< DArrayAnimation > DArrayAnimationController`

### **8.40.1 Typedef Documentation**

### 8.40.1.1 CLLAnimationController

```
typedef Animation::AnimationController< CLLAnimation > CLLAnimationController
```

### 8.40.1.2 DArrayAnimationController

```
typedef Animation::AnimationController< DArrayAnimation > DArrayAnimationController
```

### 8.40.1.3 DLLAnimationController

```
typedef Animation::AnimationController< DLLAnimation > DLLAnimationController
```

### 8.40.1.4 SLLAnimationController

```
typedef Animation::AnimationController< SLLAnimation > SLLAnimationController
```

## 8.41 AnimationController.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef CORE_ANIMATION_ANIMATIONCONTROLLER_HPP
00002 #define CORE_ANIMATION_ANIMATIONCONTROLLER_HPP
00003
00004 #include "AnimationState.hpp"
00005 #include "Container.hpp"
00006 #include "Core/Deque.hpp"
00007
00008 namespace Animation {
00009     template< typename T = SLLAnimation >
00010     class AnimationController {
00011     public:
00012         typedef std::shared_ptr< AnimationController > Ptr;
00013
00014     private:
00015         std::vector< T > animationGroup;
00016
00017         static constexpr float defaultSpeed = 1;
00018
00019     public:
00020         std::size_t mCurrentAnimationIndex;
00021
00022         float mSpeed;
00023
00024         bool mPlaying;
00025
00026         bool mInteractionLock;
00027
00028         static constexpr float stopDuration = 0.25;
00029
00030         float mCurrStopDuration;
00031
00032     public:
00033         AnimationController();
00034
00035         ~AnimationController();
00036
00037         void RunAll();
```

```
00082     void Reset();
00083
00087     void ResetCurrent();
00088
00093     void SetAnimation(std::size_t animationIndex);
00094
00099     std::size_t CurrentAnimationIndex() const;
00100
00101 public:
00106     void AddAnimation(T animation);
00107
00111     void PopAnimation();
00112
00116     void Clear();
00117
00118 public:
00123     float GetAnimationDuration();
00124
00129     std::size_t GetNumAnimation() const;
00130
00135     std::size_t GetAnimationIndex() const;
00136
00141     bool Done() const;
00142
00147     bool IsPlaying() const;
00148
00149 public:
00156     void StepForward();
00157
00164     void StepBackward();
00165
00169     void Pause();
00170
00174     void Continue();
00175
00180     void InteractionLock();
00181
00187     void InteractionAllow();
00188
00193     bool IsInteractionAllow() const;
00194
00195 public:
00200     void Update(float dt);
00201
00206     void SetSpeed(float speed);
00207
00212     float GetSpeed() const;
00213
00214 public:
00219     T GetAnimation();
00220
00221 private:
00227     float GetAnimateFrame(float dt) const;
00228
00233     float GetStopDuration();
00234 };
00235 }; // namespace Animation
00236
00237 template< typename T >
00238 Animation::AnimationController< T >::AnimationController()
00239 : mSpeed(defaultSpeed), animationGroup({}), mCurrentAnimationIndex(0),
00240   mPlaying(false), mInteractionLock(false), mCurrStopDuration(0.0f) {}
00241
00242 template< typename T >
00243 Animation::AnimationController< T >::~AnimationController() {}
00244
00245 template< typename T >
00246 void Animation::AnimationController< T >::RunAll() {
00247     if (animationGroup.empty()) return;
00248     ResetCurrent();
00249     SetAnimation(animationGroup.size() - 1);
00250     animationGroup.back().PlayingAt(animationGroup.back().GetDuration());
00251 }
00252
00253 template< typename T >
00254 void Animation::AnimationController< T >::Reset() {
00255     if (!IsInteractionAllow()) return;
00256     for (auto &animation : animationGroup) {
00257         animation.Reset();
00258     }
00259     SetAnimation(0);
00260     Continue();
00261 }
00262
00263 template< typename T >
00264 void Animation::AnimationController< T >::StepForward() {
00265     if (mCurrentAnimationIndex == animationGroup.size() - 1) {
```

```
00266     RunAll();
00267     return;
00268 }
00269 ResetCurrent();
00270 SetAnimation(mCurrentAnimationIndex + 1);
00271 ResetCurrent();
00272 }
00273
00274 template< typename T >
00275 void Animation::AnimationController< T >::StepBackward() {
00276     ResetCurrent();
00277     SetAnimation(mCurrentAnimationIndex - 1);
00278     ResetCurrent();
00279 }
00280
00281 template< typename T >
00282 void Animation::AnimationController< T >::SetAnimation(std::size_t index) {
00283     if (!IsInteractionAllow()) return;
00284     if (0 <= index && index < GetNumAnimation()) {
00285         if (mCurrentAnimationIndex > index) {
00286             for (; mCurrentAnimationIndex > index; mCurrentAnimationIndex--) {
00287                 ResetCurrent();
00288             }
00289         }
00290         mCurrentAnimationIndex = index;
00291     }
00292 }
00293
00294 template< typename T >
00295 std::size_t Animation::AnimationController< T >::CurrentAnimationIndex() const {
00296     return mCurrentAnimationIndex;
00297 }
00298
00299 template< typename T >
00300 void Animation::AnimationController< T >::AddAnimation(T animation) {
00301     animationGroup.push_back(animation);
00302 }
00303
00304 template< typename T >
00305 void Animation::AnimationController< T >::PopAnimation() {
00306     assert(GetNumAnimation() > 0);
00307     animationGroup.pop_back();
00308 }
00309
00310 template< typename T >
00311 inline void Animation::AnimationController< T >::Clear() {
00312     animationGroup.clear();
00313 }
00314
00315 template< typename T >
00316 void Animation::AnimationController< T >::Pause() {
00317     if (IsInteractionAllow()) mPlaying = false;
00318 }
00319
00320 template< typename T >
00321 void Animation::AnimationController< T >::ResetCurrent() {
00322     if (!IsInteractionAllow()) return;
00323     if (animationGroup.empty()) return;
00324     animationGroup[mCurrentAnimationIndex].Reset();
00325 }
00326
00327 template< typename T >
00328 void Animation::AnimationController< T >::Continue() {
00329     if (IsInteractionAllow()) mPlaying = true;
00330 }
00331
00332 template< typename T >
00333 inline void Animation::AnimationController< T >::InteractionLock() {
00334     mInteractionLock = true;
00335 }
00336
00337 template< typename T >
00338 inline void Animation::AnimationController< T >::InteractionAllow() {
00339     mInteractionLock = false;
00340 }
00341
00342 template< typename T >
00343 inline bool Animation::AnimationController< T >::IsInteractionAllow() const {
00344     return !mInteractionLock;
00345 }
00346
00347 template< typename T >
00348 float Animation::AnimationController< T >::GetAnimationDuration() {
00349     float totalDuration = 0.0f;
00350     for (auto &animation : animationGroup) {
00351         totalDuration += animation.GetDuration();
00352     }
```

```

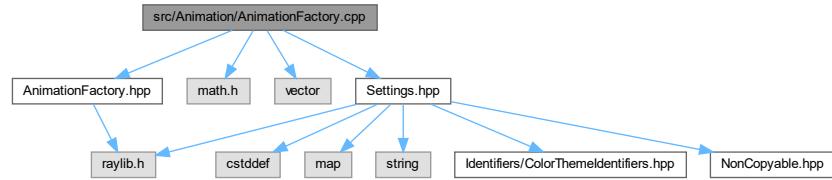
00353     return totalDuration;
00354 }
00355
00356 template< typename T >
00357 void Animation::AnimationController< T >::Update(float dt) {
00358     if (animationGroup.empty()) return;
00359     dt = GetAnimateFrame(dt);
00360     animationGroup[mCurrentAnimationIndex].Update(dt);
00361     if (IsPlaying() && animationGroup[mCurrentAnimationIndex].Done()) {
00362         // std::cout << dt << " | " << mCurrentAnimationIndex << " - "
00363         // << GetStopDuration() << " " << stopDuration << std::endl;
00364         if (GetStopDuration() >= stopDuration) {
00365             SetAnimation(mCurrentAnimationIndex + 1);
00366             mCurrStopDuration = 0.0f;
00367         } else {
00368             mCurrStopDuration = mCurrStopDuration + dt;
00369         }
00370     }
00371     if (Done()) Pause();
00372 }
00373
00374 template< typename T >
00375 void Animation::AnimationController< T >::SetSpeed(float speed) {
00376     mSpeed = speed;
00377 }
00378
00379 template< typename T >
00380 float Animation::AnimationController< T >::GetSpeed() const {
00381     return mSpeed;
00382 }
00383
00384 template< typename T >
00385 T Animation::AnimationController< T >::GetAnimation() {
00386     if (animationGroup.empty()) return T();
00387     return animationGroup[mCurrentAnimationIndex];
00388 }
00389
00390 template< typename T >
00391 inline float Animation::AnimationController< T >::GetStopDuration() {
00392     return mCurrStopDuration;
00393 }
00394
00395 template< typename T >
00396 float Animation::AnimationController< T >::GetAnimateFrame(float dt) const {
00397     if (!IsPlaying() || animationGroup.empty() || Done()) return 0.0f;
00398     return dt * mSpeed;
00399 }
00400
00401 template< typename T >
00402 std::size_t Animation::AnimationController< T >::GetNumAnimation() const {
00403     return animationGroup.size();
00404 }
00405
00406 template< typename T >
00407 std::size_t Animation::AnimationController< T >::GetAnimationIndex() const {
00408     return mCurrentAnimationIndex;
00409 }
00410
00411 template< typename T >
00412 bool Animation::AnimationController< T >::Done() const {
00413     if (animationGroup.empty()) return true;
00414     return animationGroup.back().Done();
00415 }
00416
00417 template< typename T >
00418 bool Animation::AnimationController< T >::IsPlaying() const {
00419     return mPlaying;
00420 }
00421
00422 /* */
00423 typedef Animation::AnimationController< SLLAnimation > SLLAnimationController;
00424 typedef Animation::AnimationController< DLLAnimation > DLLAnimationController;
00425 typedef Animation::AnimationController< CLLAnimation > CLLAnimationController;
00426 typedef Animation::AnimationController< DArrayAnimation >
00427     DArrayAnimationController;
00428
00429 #endif // CORE_ANIMATION_ANIMATIONCONTROLLER_HPP

```

## 8.42 src/Animation/AnimationFactory.cpp File Reference

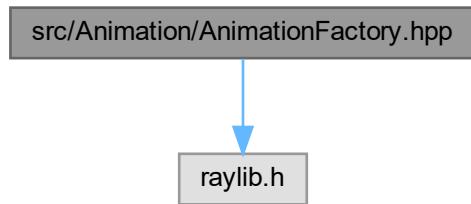
```
#include "AnimationFactory.hpp"
#include <math.h>
```

```
#include <vector>
#include "Settings.hpp"
Include dependency graph for AnimationFactory.cpp:
```

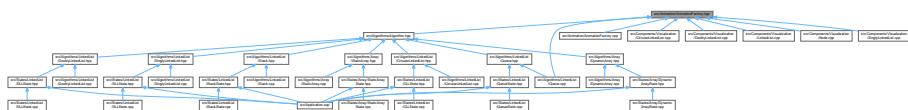


## 8.43 src/Animation/AnimationFactory.hpp File Reference

```
#include "raylib.h"
Include dependency graph for AnimationFactory.hpp:
```



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



# Namespaces

- namespace AnimationFactory

The animation factory namespace, contains all the animation helper functions.

## Functions

- float `AnimationFactory::BounceOut` (float t)
- float `AnimationFactory::ElasticOut` (float t)
- void `AnimationFactory::DrawDirectionalArrow` (Vector2 start, Vector2 end, bool active, float t)
- void `AnimationFactory::DrawActiveArrow` (Vector2 start, Vector2 end, float t)
- void `AnimationFactory::DrawDoubleDirectionalArrow` (Vector2 start, Vector2 end, bool activeStart, bool activeEnd, float tStart, float tEnd)
- void `AnimationFactory::DrawDoubleActiveArrow` (Vector2 start, Vector2 end, float tStart, float tEnd)
- void `AnimationFactory::DrawCircularArrow` (Vector2 start, Vector2 end, bool active, float t)
- float `AnimationFactory::Dist` (Vector2 p1, Vector2 p2)
- Vector2 `AnimationFactory::InverseVector` (Vector2 vector)
- Vector2 `AnimationFactory::MoveNode` (Vector2 src, Vector2 dst, float t)
- Color `AnimationFactory::BlendColor` (Color src, Color dst, float t)
- void `AnimationFactory::ReCalculateEnds` (Vector2 &start, Vector2 &end, float radius, bool applyX=true, bool applyY=true)

## 8.44 AnimationFactory.hpp

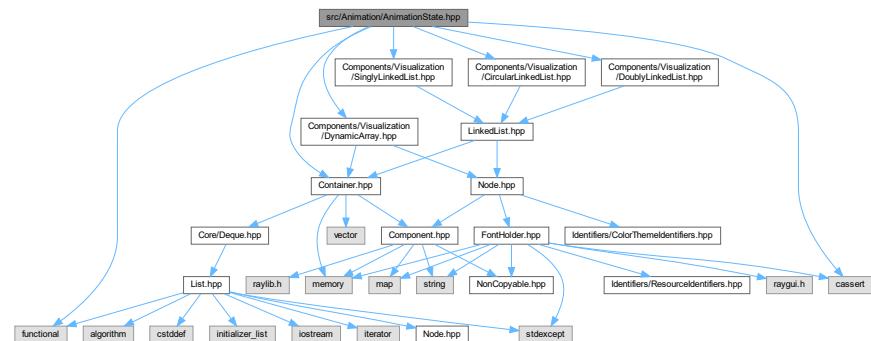
[Go to the documentation of this file.](#)

```
00001 #ifndef CORE_ANIMATION_ANIMATIONFACTORY_HPP
00002 #define CORE_ANIMATION_ANIMATIONFACTORY_HPP
00003
00004 #include "raylib.h"
00005
00010 namespace AnimationFactory {
00011     float BounceOut(float t);
00012     float ElasticOut(float t);
00013     void DrawDirectionalArrow(Vector2 start, Vector2 end, bool active, float t);
00014     void DrawActiveArrow(Vector2 start, Vector2 end, float t);
00015
00016     void DrawDoubleDirectionalArrow(Vector2 start, Vector2 end,
00017                                     bool activeStart, bool activeEnd,
00018                                     float tStart, float tEnd);
00019
00020     void DrawDoubleActiveArrow(Vector2 start, Vector2 end, float tStart,
00021                               float tEnd);
00022
00023     void DrawCircularArrow(Vector2 start, Vector2 end, bool active, float t);
00024
00025     float Dist(Vector2 p1, Vector2 p2);
00026     Vector2 InverseVector(Vector2 vector);
00027     Vector2 MoveNode(Vector2 src, Vector2 dst, float t);
00028
00029     Color BlendColor(Color src, Color dst, float t);
00030
00031     void ReCalculateEnds(Vector2 &start, Vector2 &end, float radius,
00032                           bool applyX = true, bool applyY = true);
00033     /* Empty AnimationState */
00034 }; // namespace AnimationFactory
00035
00036 #endif // CORE_ANIMATION_ANIMATIONFACTORY_HPP
```

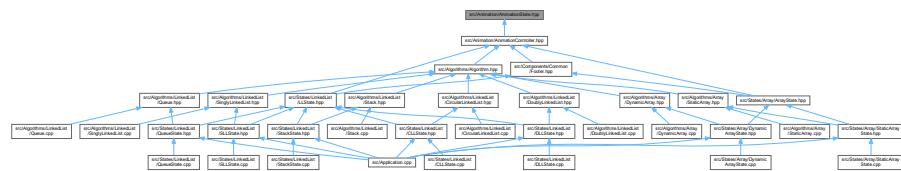
## 8.45 src/Animation/AnimationState.hpp File Reference

```
#include <functional>
#include "Components/Visualization/CircularLinkedList.hpp"
#include "Components/Visualization/DoublyLinkedList.hpp"
#include "Components/Visualization/DynamicArray.hpp"
#include "Components/Visualization/SinglyLinkedList.hpp"
#include "Container.hpp"
```

```
#include <cassert>
Include dependency graph for AnimationState.hpp:
```



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



## Classes

- class Animation::AnimationState< T >  
*The animation state class.*

# Namespaces

- namespace Animation

## TypeDefs

- `typedef Animation::AnimationState< GUIVisualizer::SinglyLinkedList > SLLAnimation`
  - `typedef Animation::AnimationState< GUIVisualizer::DoublyLinkedList > DLLAnimation`
  - `typedef Animation::AnimationState< GUIVisualizer::CircularLinkedList > CLLAnimation`
  - `typedef Animation::AnimationState< GUIVisualizer::DynamicArray > DArrayAnimation`

## 8.45.1 Typedef Documentation

### 8.45.1.1 CLLAnimation

```
typedef Animation::AnimationState< GUIVisualizer::CircularLinkedList > CLLAnimation
```

### 8.45.1.2 DArrayAnimation

```
typedef Animation::AnimationState< GUIVisualizer::DynamicArray > DArrayAnimation
```

### 8.45.1.3 DLLAnimation

```
typedef Animation::AnimationState< GUIVisualizer::DoublyLinkedList > DLLAnimation
```

### 8.45.1.4 SLLAnimation

```
typedef Animation::AnimationState< GUIVisualizer::SinglyLinkedList > SLLAnimation
```

## 8.46 AnimationState.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef CORE_ANIMATION_ANIMATIONSTATE_HPP
00002 #define CORE_ANIMATION_ANIMATIONSTATE_HPP
00003
00004 #include <functional>
00005
00006 #include "Components/Visualization/CircularLinkedList.hpp"
00007 #include "Components/Visualization/DoublyLinkedList.hpp"
00008 #include "Components/Visualization/DynamicArray.hpp"
00009 #include "Components/Visualization/SinglyLinkedList.hpp"
00010 #include "Container.hpp"
00011
00012 namespace Animation {
00013     template< typename T >
00014     class AnimationState {
00015     private:
00016         float mDuration;
00017
00018         float mCurrentPlayingAt;
00019
00020         int mHighlightedLine;
00021
00022         std::string actionDescription;
00023
00024     private:
00025         T mDataStructureBefore;
00026
00027         std::function< T(T, float, Vector2) > mAnimation;
00028
00029     public:
00030         AnimationState();
00031
00032         ~AnimationState();
00033
00034         void PlayingAt(float playingAt);
00035
00036         float GetCurrentPlayingAt() const;
00037
00038         void Draw(Vector2 base = (Vector2){0, 0});
00039
00040     };
00041 }
```

```
00082         void Update(float dt);
00083
00084     void Reset();
00085
00086 public:
00087     void SetDuration(float duration);
00088
00089     float GetDuration() const;
00090
00091     void SetAnimation(std::function< T(T, float, Vector2) > animation);
00092
00093     void SetSourceDataStructure(T dataStructure);
00094
00095     T GetDataStructure(float progress, Vector2 base = (Vector2){0, 0});
00096
00097 public:
00098     bool Done() const;
00099
00100    void SetHighlightLine(int line);
00101
00102    int GetHighlightedLine() const;
00103
00104    void SetActionDescription(std::string description);
00105
00106    std::string GetActionDescription() const;
00107 };
00108 }; // namespace Animation
00109
00110 #include <cassert>
00111
00112 template< typename T >
00113 Animation::AnimationState< T >::AnimationState()
00114     : mCurrentPlayingAt(0.0f), mDuration(0.0f), mHighlightedLine(-1),
00115       mDataStructureBefore(nullptr), actionDescription("") {
00116     mAnimation = [] (T srcDS, float playingAt, Vector2 base) {
00117         srcDS.Draw(base, playingAt);
00118         return srcDS;
00119     };
00120 }
00121
00122 template< typename T >
00123 Animation::AnimationState< T >::~AnimationState() {}
00124
00125 template< typename T >
00126 void Animation::AnimationState< T >::SetDuration(float duration) {
00127     mDuration = duration;
00128 }
00129
00130 template< typename T >
00131 float Animation::AnimationState< T >::GetDuration() const {
00132     return mDuration;
00133 }
00134
00135 template< typename T >
00136 void Animation::AnimationState< T >::SetAnimation(
00137     std::function< T(T, float, Vector2) > animation) {
00138     mAnimation = animation;
00139 }
00140
00141 template< typename T >
00142 void Animation::AnimationState< T >::PlayingAt(float playingAt) {
00143     mCurrentPlayingAt = playingAt;
00144 }
00145
00146 template< typename T >
00147 float Animation::AnimationState< T >::GetCurrentPlayingAt() const {
00148     return mCurrentPlayingAt;
00149 }
00150
00151 template< typename T >
00152 void Animation::AnimationState< T >::SetSourceDataStructure(T dataStructure) {
00153     mDataStructureBefore = dataStructure;
00154 }
00155
00156 template< typename T >
00157 T Animation::AnimationState< T >::GetDataStructure(float progress,
00158                                                       Vector2 base) {
00159     assert(progress >= 0.0f && progress <= 1.0f);
00160     return mAnimation(mDataStructureBefore, progress, base);
00161 }
00162
00163 template< typename T >
00164 void Animation::AnimationState< T >::Update(float dt) {
00165     mCurrentPlayingAt += dt;
00166     if (mCurrentPlayingAt > mDuration + 1.0f)
00167         mCurrentPlayingAt = mDuration + 1.0f;
```

```

00218 }
00219
00220 template< typename T >
00221 void Animation::AnimationState< T >::Reset() {
00222     PlayingAt(0.0f);
00223 }
00224
00225 template< typename T >
00226 bool Animation::AnimationState< T >::Done() const {
00227     return mCurrentPlayingAt >= mDuration;
00228 }
00229
00230 template< typename T >
00231 void Animation::AnimationState< T >::SetHighlightedLine(int line) {
00232     mHighlightedLine = line;
00233 }
00234
00235 template< typename T >
00236 int Animation::AnimationState< T >::GetHighlightedLine() const {
00237     return mHighlightedLine;
00238 }
00239
00240 template< typename T >
00241 inline void Animation::AnimationState< T >::SetActionDescription(
00242     std::string description) {
00243     actionDescription = description;
00244 }
00245
00246 template< typename T >
00247 inline std::string Animation::AnimationState< T >::GetActionDescription()
00248     const {
00249     return actionDescription;
00250 }
00251
00252 template< class T >
00253 void Animation::AnimationState< T >::Draw(Vector2 base) {
00254     float playingAt = std::min(
00255         1.0f, (mDuration == 0.0f ? 1.0f : mCurrentPlayingAt / mDuration));
00256     mDataStructureBefore, playingAt, base);
00257 }
00258
00259 typedef Animation::AnimationState< GUIVisualizer::SinglyLinkedList >
00260     SLLAnimation;
00261 typedef Animation::AnimationState< GUIVisualizer::DoublyLinkedList >
00262     DLLAnimation;
00263 typedef Animation::AnimationState< GUIVisualizer::CircularLinkedList >
00264     CLLAnimation;
00265 typedef Animation::AnimationState< GUIVisualizer::DynamicArray >
00266     DArrayAnimation;
00267
00268 #endif // CORE_ANIMATION_ANIMATIONSTATE_HPP

```

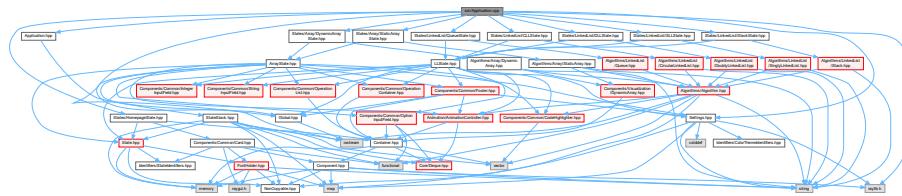
## 8.47 src/Application.cpp File Reference

```

#include "Application.hpp"
#include "raylib.h"
#include "Global.hpp"
#include "Settings.hpp"
#include "States/Array/DynamicArrayState.hpp"
#include "States/Array/StaticArrayState.hpp"
#include "States/HomepageState.hpp"
#include "States/LinkedList/CLLState.hpp"
#include "States/LinkedList/DLLState.hpp"
#include "States/LinkedList/QueueState.hpp"
#include "States/LinkedList/SLLState.hpp"
#include "States/LinkedList/StackState.hpp"

```

Include dependency graph for Application.cpp:



## Macros

- `#define RAYGUI_IMPLEMENTATION`

### 8.47.1 Macro Definition Documentation

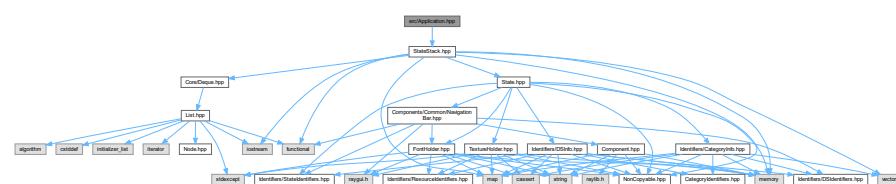
#### 8.47.1.1 RAYGUI\_IMPLEMENTATION

```
#define RAYGUI_IMPLEMENTATION
```

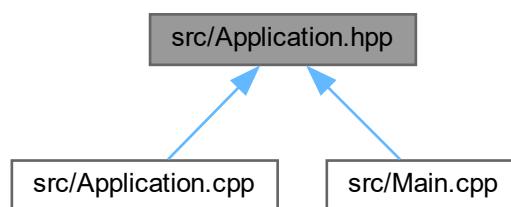
## 8.48 src/Application.hpp File Reference

```
#include "StateStack.hpp"
```

Include dependency graph for Application.hpp:



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



## Classes

- class [Application](#)

*The application class that represents the application.*

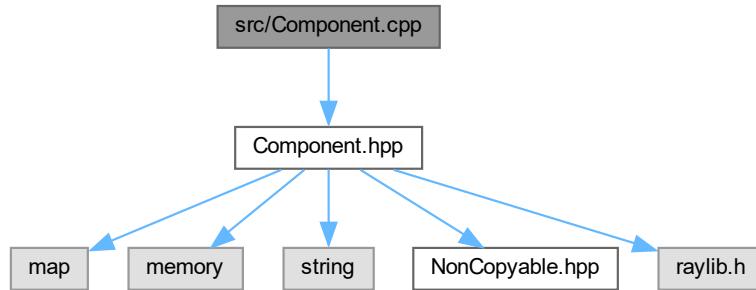
## 8.49 Application.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef APPLICATION_HPP
00002 #define APPLICATION_HPP
00003
00004 #include "StateStack.hpp"
00005
00011 class Application {
00012 public:
00016     Application();
00017     ~Application();
00022
00027     void Run();
00028
00033     void Close();
00034
00035     void Init();
00036
00042     bool WindowClosed();
00043
00044 private:
00049     void Render();
00050
00056     void RegisterStates();
00057
00064     void LoadResources();
00065
00071     void Update(float dt);
00072
00073 private:
00074     bool closed = false;
00075     // Image favicon;
00076
00077 private:
00084     State::StateStack mStack;
00085
00091     FontHolder* fonts;
00092
00099     TextureHolder* images;
00100
00107     CategoryInfo* categories;
00108
00115     DSInfo* dsInfo;
00116 };
00117
00118 #endif // APPLICATION_HPP
```

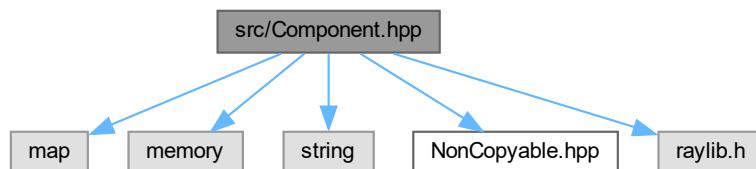
## 8.50 src/Component.cpp File Reference

```
#include "Component.hpp"
Include dependency graph for Component.cpp:
```

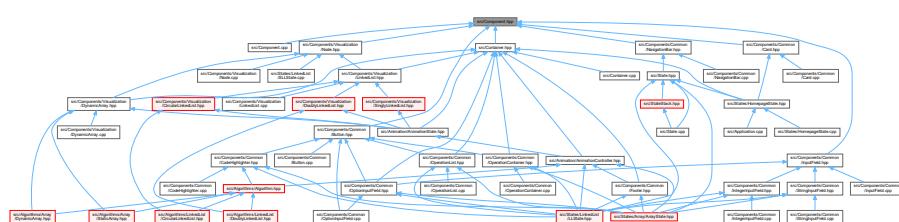


## 8.51 src/Component.hpp File Reference

```
#include <map>
#include <memory>
#include <string>
#include "NonCopyable.hpp"
#include "raylib.h"
Include dependency graph for Component.hpp:
```



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



## Classes

- class [GUI::Component](#)

*The base component class that is used to represent a [GUI](#) component in the [GUI](#).*

## Namespaces

- namespace [GUI](#)

## 8.52 Component.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef COMPONENT_HPP
00002 #define COMPONENT_HPP
00003
00004 #include <map>
00005 #include <memory>
00006 #include <string>
00007
00008 #include "NonCopyable.hpp"
00009 #include "raylib.h"
00010
00011 namespace GUI {
00012     class Component
00013     //: private NonCopyable< Component >
00014     {
00015         public:
00016             typedef std::shared_ptr< Component > Ptr;
00017
00018         public:
00019             Component();
00020
00021             virtual ~Component();
00022
00023             virtual void Draw(Vector2 base = (Vector2){0, 0});
00024
00025             virtual bool IsSelectable() const = 0;
00026
00027             bool isSelected() const;
00028
00029             virtual void Select();
00030
00031             virtual void Deselect();
00032
00033             virtual void ToggleVisible();
00034
00035             virtual void SetVisible(bool visible);
00036
00037             virtual bool GetVisible();
00038
00039
00040         public:
00041             void SetPosition(float x, float y);
00042
00043             void SetPosition(Vector2 position);
00044
00045             Vector2GetPosition();
00046
00047             virtual Vector2 GetSize();
00048
00049     protected:
00050         Vector2 mPosition;
00051
00052         bool mVisible;
00053
00054     private:
00055         bool mIsSelected;
00056
00057     protected:
00058         virtual void UpdateMouseCursorWhenHover(
00059             std::map< std::string, Rectangle > bounds, bool hover,
00060             bool noHover);
00061
00062         virtual void UpdateMouseCursorWhenHover(Rectangle bound, bool hover,
00063                                         bool noHover);
00064
00065
00066
```

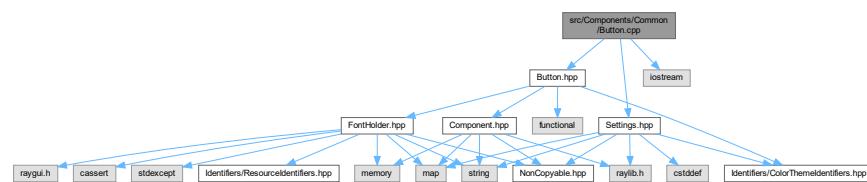
```

00172     virtual bool GetHoverStatus(std::map< std::string, Rectangle > bounds,
00173                                 bool hover, bool noHover);
00174
00180     virtual bool GetHoverStatus(Rectangle bound, bool hover, bool noHover);
00181 };
00182 // namespace GUI
00183
00184 #endif // COMPONENT_HPP

```

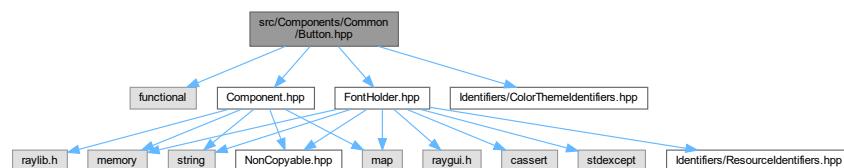
## 8.53 src/Components/Common/Button.cpp File Reference

```
#include "Button.hpp"
#include <iostream>
#include "Settings.hpp"
Include dependency graph for Button.cpp:
```



## 8.54 src/Components/Common/Button.hpp File Reference

```
#include <functional>
#include "Component.hpp"
#include "FontHolder.hpp"
#include "Identifiers/ColorThemeIdentifiers.hpp"
Include dependency graph for Button.hpp:
```



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



## Classes

- class [GUIComponent::Button](#)

*The button class that is used to represent a button in the GUI.*

## Namespaces

- namespace [GUIComponent](#)

## 8.55 Button.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef COMPONENTS_Guibutton_HPP
00002 #define COMPONENTS_Guibutton_HPP
00003
00004 #include <functional>
00005
00006 #include "Component.hpp"
00007 #include "FontHolder.hpp"
00008 #include "Identifiers/ColorThemeIdentifiers.hpp"
00009
00010 namespace GUIComponent {
00011     class Button : public GUI::Component {
00012     public:
00013         enum TextAlignment { Left, Center, Right, AlignmentCount };
00014         typedef std::shared_ptr< Button > Ptr;
00015         Button(std::string text, FontHolder* fonts);
00016         Button();
00017         ~Button();
00018         void Draw(Vector2 base = (Vector2){0, 0});
00019
00020         void SetAction(std::function< void() > clickedAction);
00021
00022         bool IsSelectable() const;
00023
00024         void SetButtonHoverColor(ColorTheme::ID color);
00025         void SetButtonColor(ColorTheme::ID color);
00026         void SetTextColor(ColorTheme::ID color);
00027         void SetSize(float width, float height);
00028         voidSetText(std::string text);
00029
00030         void SetFontSize(float textSize);
00031         float GetFontSize() const;
00032         void SetTextAlignment(TextAlignment textAlignment);
00033
00034         void EnableFitContent();
00035         void DisableFitContent();
00036
00037         Vector2 GetSize();
00038
00039         void SetActionOnHover(bool actionOnHover);
00040
00041         void FitContent();
00042
00043         void SetContentPos();
00044
00045         void SetContentSize();
00046
00047         bool IsClicked();
00048         void DrawButtonText();
00049         Vector2 GetContentPos();
00050         Vector2 GetContentSize();
00051
00052         void FitContent();
00053
00054         private:
00055             FontHolder* fonts;
00056             std::string content;
00057             ColorTheme::ID buttonColorTheme;
00058             ColorTheme::ID buttonHoverColorTheme;
00059             ColorTheme::ID textColorTheme;
00060             TextAlignment alignment;
00061             float fontSize;
00062
00063         private:
00064             bool fitContent;
00065             bool mActionOnHover;
00066

```

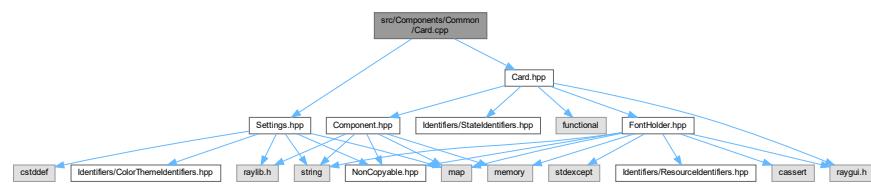
```

00067     private:
00068         bool isHover;
00069         Rectangle bound;
00070         std::function< void() > action;
00071     };
00072 }; // namespace GUIComponent
00073
00074 #endif // COMPONENTS_Guibutton_HPP

```

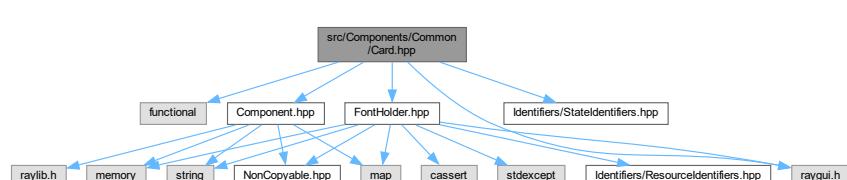
## 8.56 src/Components/Common/Card.cpp File Reference

```
#include "Card.hpp"
#include "Settings.hpp"
Include dependency graph for Card.cpp:
```

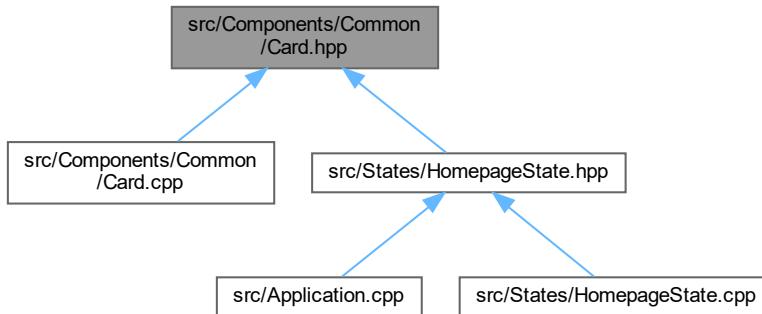


## 8.57 src/Components/Common/Card.hpp File Reference

```
#include <functional>
#include "Component.hpp"
#include "FontHolder.hpp"
#include "Identifiers/StateIdentifiers.hpp"
#include "raygui.h"
Include dependency graph for Card.hpp:
```



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



## Classes

- class [GUIComponent::Card](#)  
*The card class that is used to represent a card in the GUI.*

## Namespaces

- namespace [GUIComponent](#)

## 8.58 Card.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef COMPONENTS_CARD_HPP
00002 #define COMPONENTS_CARD_HPP
00003
00004 #include <functional>
00005
00006 #include "Component.hpp"
00007 #include "FontHolder.hpp"
00008 #include "Identifiers/StateIdentifiers.hpp"
00009 #include "raygui.h"
0010
0011 /* Little note
0012     - Card height: 250
0013     - Card width: 200
0014     - Title: 24px
0015     - Image: 160x250
0016
0017 */
0018
0019 namespace GUIComponent {
0020     class Card : public GUI::Component {
0021     public:
0022         Card(std::string text, Texture thumbnail, FontHolder* fonts);
0023         Card();
0024         ~Card();
0025         void SetLink(std::function< void(States::ID) > link);
0026         void SetStateID(States::ID stateID);
0027
0028         void SetText(std::string text);
0029
0030         bool IsSelectable() const;
0031         void Draw(Vector2 base = (Vector2){0, 0});
0032
0033     public:
0034         void Draw();
0035     };
0036 }
```

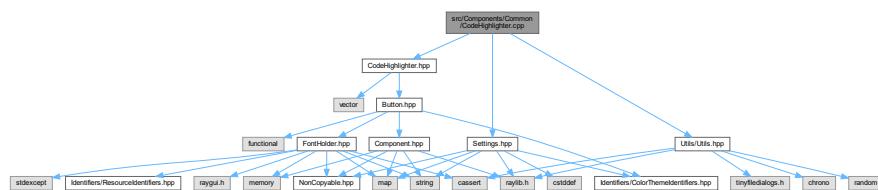
```

00037     private:
00038         bool DrawImage(Vector2 base);
00039         bool DrawTitle(Vector2 base);
00040
00041     private:
00042         FontHolder* fonts;
00043         Texture thumbnail;
00044         std::string title;
00045         std::function< void(States::ID) > toLink;
00046         States::ID stateID;
00047
00048     private:
00049         std::map< std::string, Rectangle > hoverBounds;
00050         bool isHover;
00051     };
00052 }; // namespace GUIComponent
00053
00054 #endif // COMPONENTS_CARD_HPP

```

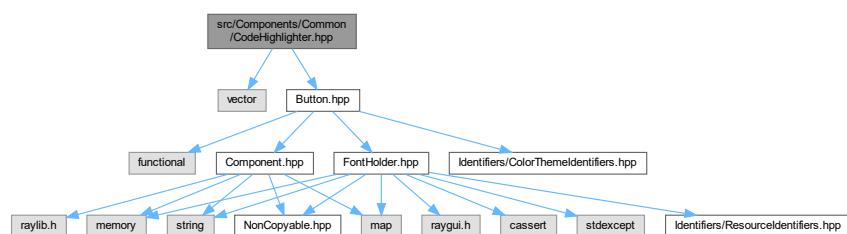
## 8.59 src/Components/Common/CodeHighlighter.cpp File Reference

```
#include "CodeHighlighter.hpp"
#include "Settings.hpp"
#include "Utils/Utils.hpp"
Include dependency graph for CodeHighlighter.cpp:
```

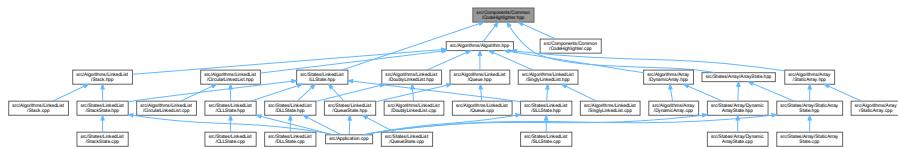


## 8.60 src/Components/Common/CodeHighlighter.hpp File Reference

```
#include <vector>
#include "Button.hpp"
Include dependency graph for CodeHighlighter.hpp:
```



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



## Classes

- class `GUIComponent::CodeHighlighter`

*The code highlighter class that is used to represent a code highlighter in the GUI.*

## Namespaces

- namespace **GUIComponent**

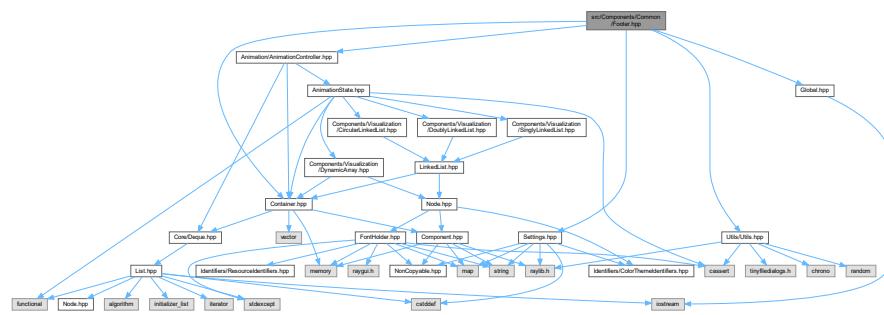
## 8.61 CodeHighlighter.hpp

[Go to the documentation of this file.](#)

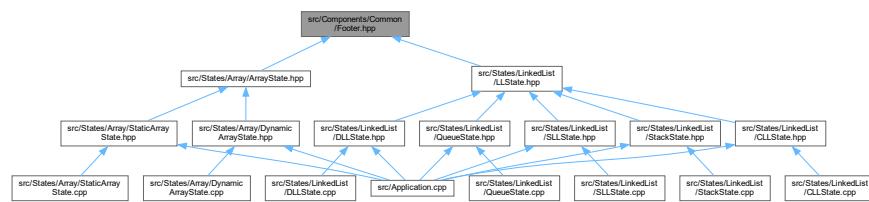
```
00001 #ifndef COMPONENTS_COMMON_CODEHIGHLIGHTER_HPP
00002 #define COMPONENTS_COMMON_CODEHIGHLIGHTER_HPP
00003
00004 #include <vector>
00005
00006 // #include "Component.hpp"
00007 #include "Button.hpp"
00008
00009 namespace GUIComponent {
00010     class CodeHighlighter : public GUI::Component {
00011     public:
00012         typedef std::shared_ptr<CodeHighlighter> Ptr;
00013
00014     private:
00015         GUIComponent::Button mButtonShowCode;
00016         bool mShowCode;
00017
00018         GUIComponent::Button mButtonShowAction;
00019         bool mShowActionDescription;
00020
00021         FontHolder* fonts;
00022
00023     private:
00024         std::vector<std::string> mCode;
00025         int mHighlightedLine;
00026         std::string mActionDescription;
00027
00028     public:
00029         CodeHighlighter(FontHolder* fonts);
00030         ~CodeHighlighter();
00031         void Draw(Vector2 base = (Vector2){0, 0});
00032         bool IsSelectable() const;
00033         void InitButtons();
00034
00035     public:
00036         void AddCode(std::vector<std::string> code);
00037         void Highlight(int line);
00038         void ToggleShowCode();
00039         void ToggleShowAction();
00040         void SetShowCode(bool show);
00041         void SetShowAction(bool show);
00042
00043     public:
00044         void AddActionDescription(std::string description);
00045
00046     private:
00047         void DrawActionDescription(Vector2 base = (Vector2){0, 0});
00048         void DrawCodeHighlighter(Vector2 base = (Vector2){0, 0});
00049     };
00050 }; // namespace GUIComponent
00051
00052 #endif // COMPONENTS_COMMON_CODEHIGHLIGHTER_HPP
```

## 8.62 src/Components/Common/Footer.hpp File Reference

```
#include "Animation/AnimationController.hpp"
#include "Container.hpp"
#include "Global.hpp"
#include "Settings.hpp"
#include "Utils/Utils.hpp"
Include dependency graph for Footer.hpp:
```



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



## Classes

- class [GUIComponent::Footer< T >](#)  
*The footer class that is used to represent a footer in the [GUI](#).*

## Namespaces

- namespace [GUIComponent](#)

## 8.63 Footer.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef COMPONENTS_COMMON_FOOTER_HPP
00002 #define COMPONENTS_COMMON_FOOTER_HPP
00003
00004 #include "Animation/AnimationController.hpp"
00005 #include "Container.hpp"
00006 #include "Global.hpp"
00007 #include "Settings.hpp"
00008 #include "Utils/Utils.hpp"
```

```

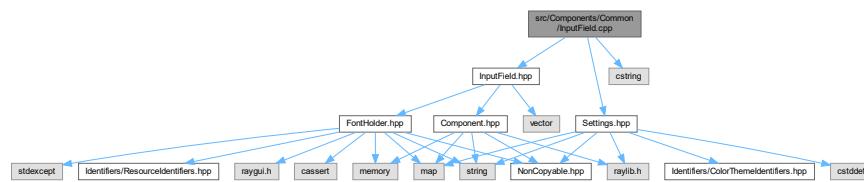
00009
00010 namespace GUIComponent {
00015     template< typename T >
00016     class Footer : public GUI::Container {
00017     public:
00018         Footer();
00019         ~Footer();
00020         void Draw(T* animController, Vector2 base = (Vector2){0, 0});
00021     };
00022 }; // namespace GUIComponent
00023
00024 template< typename T >
00025 GUIComponent::Footer< T >::Footer() {}
00026
00027 template< typename T >
00028 GUIComponent::Footer< T >::~Footer() {}
00029
00030 template< typename T >
00031 void GUIComponent::Footer< T >::Draw(T* animController, Vector2 base) {
00032     const Color backgroundColor =
00033         Settings::getInstance().getColor(ColorTheme::Footer_Background);
00034     const Color iconColor =
00035         Settings::getInstance().getColor(ColorTheme::Footer_Icon);
00036     const Color hoveredIconColor =
00037         Settings::getInstance().getColor(ColorTheme::Footer_HoveredIcon);
00038
00039     base.x += mPosition.x;
00040     base.y += mPosition.y;
00041     DrawRectangleRec((Rectangle){base.x, base.y, global::SCREEN_WIDTH, 40},
00042                     backgroundColor);
00043
00044     animController->SetSpeed(
00045         int(GuiSlider((Rectangle){base.x + 80, base.y + 15, 140, 10}, nullptr,
00046                         TextFormat("%dx", int(animController->GetSpeed())),
00047                         animController->GetSpeed(), 1, 7)));
00048
00049     { // Draw Reset
00050         if (Utils::DrawIcon(129, base.x + 345, base.y + 12, 1, iconColor,
00051                             hoveredIconColor)) {
00052             animController->Reset();
00053         }
00054     }
00055
00056     { // Draw Step Backward
00057         if (Utils::DrawIcon(130, base.x + 365, base.y + 12, 1, iconColor,
00058                             hoveredIconColor)) {
00059             animController->StepBackward();
00060         }
00061     }
00062
00063     { // Draw play button
00064         if (animController->Done()) {
00065             if (Utils::DrawIcon(74, base.x + 385, base.y + 12, 1, iconColor,
00066                             hoveredIconColor)) {
00067                 animController->Reset();
00068             }
00069         } else if (animController->IsPlaying()) {
00070             if (Utils::DrawIcon(132, base.x + 385, base.y + 12, 1, iconColor,
00071                             hoveredIconColor)) {
00072                 animController->Pause();
00073             }
00074         } else if (Utils::DrawIcon(133, base.x + 385, base.y + 12, 1, iconColor,
00075                             hoveredIconColor)) {
00076             animController->Continue();
00077         }
00078     }
00079
00080     { // Draw Step Forward
00081         if (Utils::DrawIcon(131, base.x + 405, base.y + 12, 1, iconColor,
00082                             hoveredIconColor)) {
00083             animController->StepForward();
00084         }
00085     }
00086
00087     { // Draw Run All
00088         if (Utils::DrawIcon(134, base.x + 425, base.y + 12, 1, iconColor,
00089                             hoveredIconColor)) {
00090             animController->RunAll();
00091         }
00092     }
00093
00094     // progressValue =
00095     animController->SetAnimation(
00096         GuiSliderBar((Rectangle){base.x + 450, base.y + 15, 400, 10}, nullptr,
00097                         nullptr, animController->GetAnimationIndex(), 0,
00098                         std::max(0, int(animController->GetNumAnimation() - 1))));
00099 }

```

```
00100
00101 #endif // COMPONENTS_COMMON_FOOTER_HPP
```

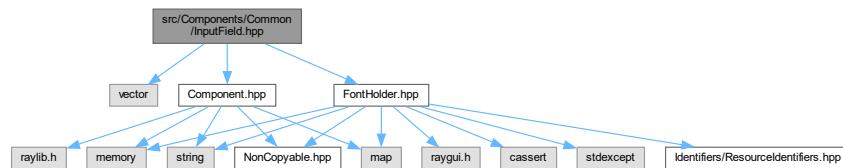
## 8.64 src/Components/Common/InputField.cpp File Reference

```
#include "InputField.hpp"
#include <cstring>
#include "Settings.hpp"
Include dependency graph for InputField.cpp:
```

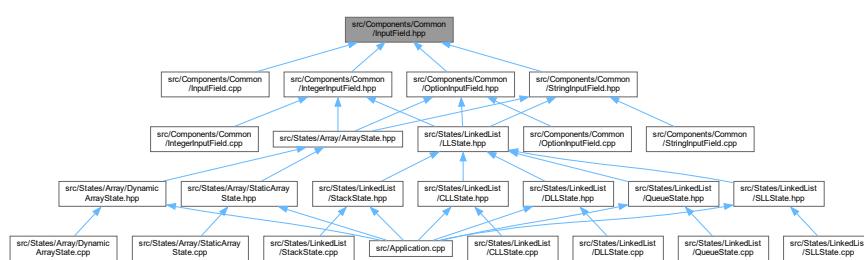


## 8.65 src/Components/Common/InputField.hpp File Reference

```
#include <vector>
#include "Component.hpp"
#include "FontHolder.hpp"
Include dependency graph for InputField.hpp:
```



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



## Classes

- class [GUIComponent::InputField](#)

*The input field class that is used to represent an input field in the GUI.*

## Namespaces

- namespace [GUIComponent](#)

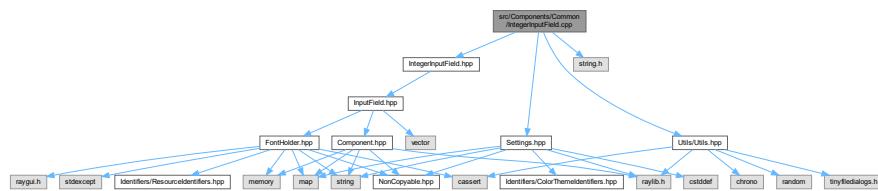
## 8.66 InputField.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef COMPONENTS_INPUTFIELD_HPP
00002 #define COMPONENTS_INPUTFIELD_HPP
00003
00004 #include <vector>
00005
00006 #include "Component.hpp"
00007 #include "FontHolder.hpp"
00008
00009 namespace GUIComponent {
00010     class InputField : public GUI::Component {
00011     public:
00012         typedef std::shared_ptr< InputField > Ptr;
00013         InputField(FontHolder* fonts);
00014         ~InputField();
00015         virtual std::string ExtractValue() = 0;
00016         virtual void Draw(Vector2 base = (Vector2){0, 0});
00017         virtual void DrawField(Vector2 base = (Vector2){0, 0}) = 0;
00018         virtual bool IsSelectable() const;
00019
00020     public:
00021         virtual void SetLabelSize(float fontSize);
00022         virtual void SetInputFieldSize(Vector2 size);
00023         virtual Vector2 GetSize();
00024         virtual void SetLabel(std::string labelContent);
00025         virtual std::string GetLabel() const;
00026
00027     protected:
00028         virtual bool IsClicked(Vector2 base = (Vector2){0, 0}) const;
00029         virtual void SetEditMode(bool canEdit);
00030         virtual bool GetEditMode() const;
00031         virtual void AllFieldDisableEdit();
00032
00033     public:
00034         virtual void Randomize();
00035
00036     protected:
00037         bool editMode;
00038         std::size_t mFieldIndex;
00039
00040     protected:
00041         float labelFontSize;
00042         Vector2 inputFieldSize;
00043
00044     protected:
00045         std::string label;
00046         FontHolder* fonts;
00047         std::string extractedValue;
00048
00049     public:
00050         static std::vector< bool > fields;
00051     };
00052 }, // namespace GUIComponent
00053
00054 #endif // COMPONENTS_INPUTFIELD_HPP
```

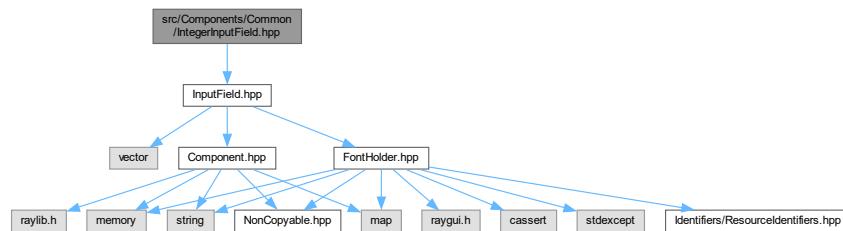
## 8.67 src/Components/Common/IntegerField.cpp File Reference

```
#include "IntegerField.hpp"
#include <string.h>
#include "Settings.hpp"
#include "Utils/Utils.hpp"
Include dependency graph for IntegerInputField.cpp:
```

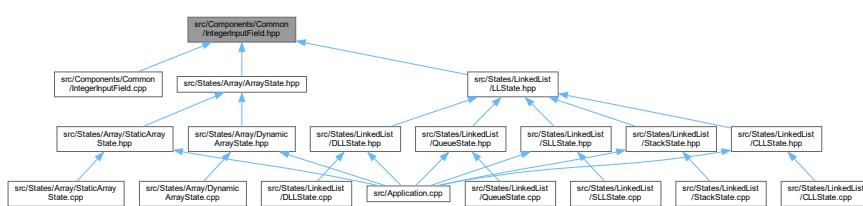


## 8.68 src/Components/Common/IntegerField.hpp File Reference

```
#include "InputField.hpp"
Include dependency graph for IntegerInputField.hpp:
```



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



## Classes

- class [GUIComponent::IntegerField](#)

*The integer input field class that is used to represent an integer input field in the GUI.*

## Namespaces

- namespace [GUIComponent](#)

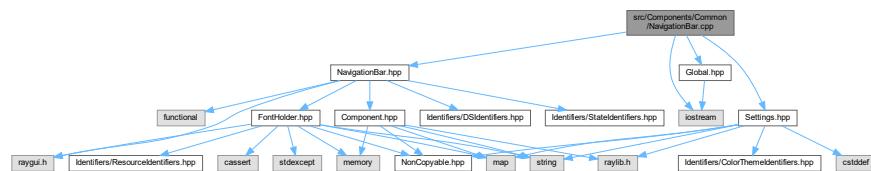
## 8.69 IntegerInputField.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef COMPONENTS_INTEGERINPUTFIELD_HPP
00002 #define COMPONENTS_INTEGERINPUTFIELD_HPP
00003
00004 #include "InputField.hpp"
00005
00006 namespace GUIComponent {
00013     class IntegerInputField : public InputField {
00014     public:
00015         typedef std::shared_ptr< IntegerInputField > Ptr;
00016         IntegerInputField(FontHolder* fonts);
00017         ~IntegerInputField();
00018         std::string ExtractValue();
00019         void DrawField(Vector2 base = (Vector2){0, 0});
00020
00021         void SetConstraint(int minValue, int maxValue);
00022         bool IsSelectable() const;
00023
00024     public:
00025         void Randomize();
00026
00027     private:
00028         int minValue;
00029         int maxValue;
00030         int input;
00031     };
00032 }, // namespace GUIComponent
00033
00034 #endif // COMPONENTS_INTEGERINPUTFIELD_HPP
```

## 8.70 src/Components/Common/NavigationBar.cpp File Reference

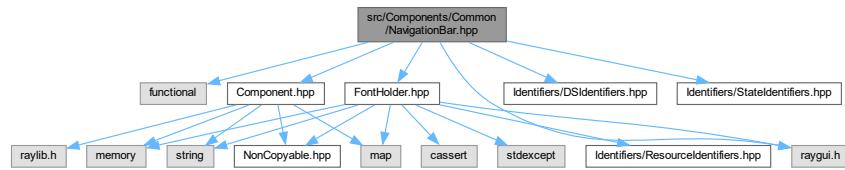
```
#include "NavigationBar.hpp"
#include <iostream>
#include "Global.hpp"
#include "Settings.hpp"
Include dependency graph for NavigationBar.cpp:
```



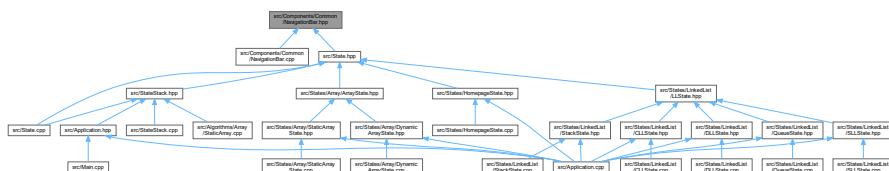
## 8.71 src/Components/Common/NavigationBar.hpp File Reference

```
#include <functional>
#include "Component.hpp"
#include "FontHolder.hpp"
#include "Identifiers/DSIdentifiers.hpp"
```

```
#include "Identifiers/StateIdentifiers.hpp"
#include "raygui.h"
Include dependency graph for NavigationBar.hpp:
```



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



## Classes

- class [GUIComponent::NavigationBar](#)  
*The navigation bar class that is used to represent a navigation bar in the GUI.*
- struct [GUIComponent::NavigationBar::TitleInfo](#)

## Namespaces

- namespace [GUIComponent](#)

## 8.72 NavigationBar.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef COMPONENTS_NAVIGATIONBAR_HPP
00002 #define COMPONENTS_NAVIGATIONBAR_HPP
00003
00004 #include <functional>
00005
00006 #include "Component.hpp"
00007 #include "FontHolder.hpp"
00008 #include "Identifiers/DSIdentifiers.hpp"
00009 #include "Identifiers/StateIdentifiers.hpp"
00010 #include "raygui.h"
0011
0012 namespace GUIComponent {
0020     class NavigationBar : public GUI::Component {
0021     public:
0022         NavigationBar(FontHolder* fonts);
0023         NavigationBar();
0024         ~NavigationBar();
0025
0026         void SetHomepageID(States::ID id);
0027         void SetDirectLink(std::function< void(States::ID) > link);
0028 }
```

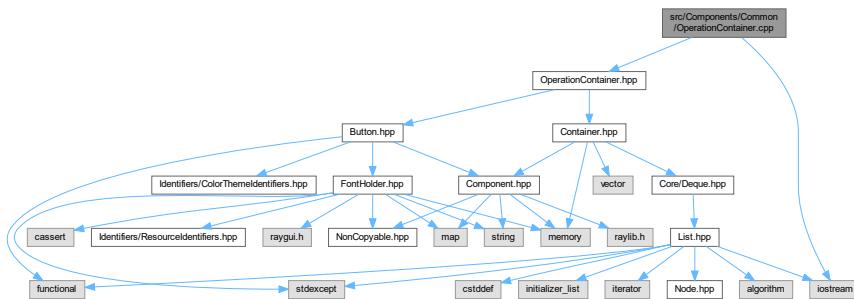
```

00029     void SetCategory(std::string category);
00030
00031     void InsertTitle(DataStructures::ID titleID, States::ID stateID,
00032                         std::string abbrTitle, std::string titleName);
00033     void SetActiveTitle(DataStructures::ID title);
00034     void ClearTitle();
00035
00036     void SetVisibleTitle(bool visible);
00037
00038 public:
00039     bool IsSelectable() const;
00040     void Draw(Vector2 base = (Vector2){0, 0});
00041
00042 private:
00043     bool DrawSwitchTheme();
00044     bool DrawLogo();
00045     States::ID DrawTitles();
00046
00047 private:
00048     struct TitleInfo {
00049         States::ID stateID;
00050         std::string abbrTitle;
00051         std::string titleName;
00052     };
00053
00054     FontHolder* fonts;
00055     std::string currentCategory;
00056     std::map< DataStructures::ID, TitleInfo > mTitles;
00057     DataStructures::ID activeTitle;
00058     bool hasTitle;
00059
00060 private:
00061     std::function< void(States::ID) > toLink;
00062     States::ID homepageID;
00063
00064 private:
00065     std::map< std::string, Rectangle > hoverBounds;
00066     bool isHover;
00067 };
00068 }; // namespace GUIComponent
00069
00070 #endif // COMPONENTS_NAVIGATIONBAR_HPP

```

## 8.73 src/Components/Common/OperationContainer.cpp File Reference

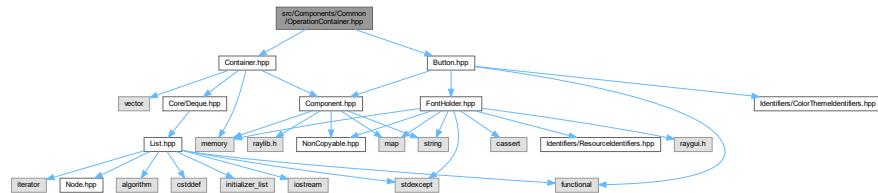
```
#include "OperationContainer.hpp"
#include <iostream>
Include dependency graph for OperationContainer.cpp:
```



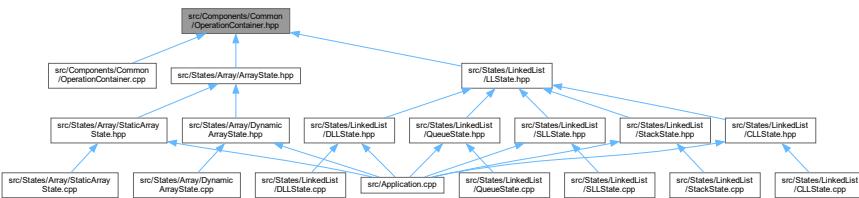
## 8.74 src/Components/Common/OperationContainer.hpp File Reference

```
#include "Button.hpp"
#include "Container.hpp"
```

Include dependency graph for OperationContainer.hpp:



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



## Classes

- class `GUIComponent::OperationContainer`

The operation container class that is used to represent an operation container in the [GUI](#).

## Namespaces

- namespace **GUIComponent**

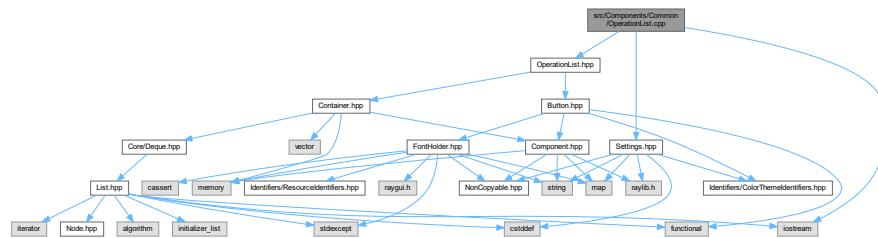
## 8.75 OperationContainer.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef COMPONENTS_OPERATIONCONTAINER_HPP
00002 #define COMPONENTS_OPERATIONCONTAINER_HPP
00003
00004 #include "Button.hpp"
00005 #include "Container.hpp"
00006
00007 namespace GUIComponent {
00015     class OperationContainer : public GUI::Container {
00016     public:
00017         typedef std::shared_ptr<OperationContainer> Ptr;
00018         OperationContainer();
00019         ~OperationContainer();
00020
00021         void DrawCurrent(Vector2 base = (Vector2){0, 0});
00022
00023         void SetVisible(bool visible);
00024
00025     private:
00026         void UpdatePosition();
00027     };
00028 }; // namespace GUIComponent
00029
00030 #endif // COMPONENTS_OPERATIONCONTAINER_HPP
```

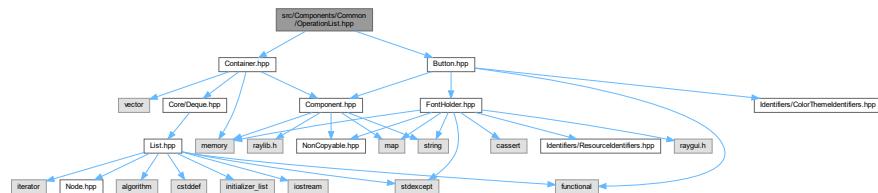
## 8.76 src/Components/Common/OperationList.cpp File Reference

```
#include "OperationList.hpp"
#include <iostream>
#include "Settings.hpp"
Include dependency graph for OperationList.cpp:
```

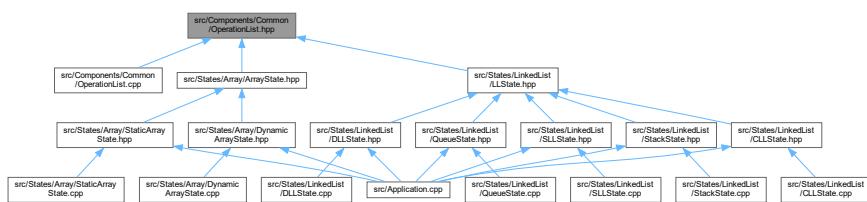


## 8.77 src/Components/Common/OperationList.hpp File Reference

```
#include "Button.hpp"
#include "Container.hpp"
Include dependency graph for OperationList.hpp:
```



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



## Classes

- class [GUIComponent::OperationList](#)

*The operation list class that is used to represent an operation list in the GUI.*

## Namespaces

- namespace [GUIComponent](#)

## 8.78 OperationList.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef COMPONENTS_OPERATIONLIST_HPP
00002 #define COMPONENTS_OPERATIONLIST_HPP
00003 #include "Button.hpp"
00004 #include "Container.hpp"
00005
00006 namespace GUIComponent {
00014   class OperationList : public GUI::Container {
00015     private:
00016       GUI::Container buttons;
00017       GUI::Container optionContainers;
00018       bool isHide;
00019       GUIComponent::Button toggleButton;
00020
00021     public:
00022       void ShowOptions(std::size_t index);
00023       void HideAllOptions();
00024       void ToggleOperations();
00025
00026     public:
00027       void Draw(Vector2 base = (Vector2){0, 0});
00028       OperationList();
00029       OperationList(FontHolder *fonts);
00030       ~OperationList();
00031       void AddOperation(GUIComponent::Button::Ptr action,
00032                         GUI::Container::Ptr optionContainer);
00033       void InitActionBar();
00034
00035       Vector2 GetSize();
00036   };
00037 }; // namespace GUIComponent
00038
00039 #endif // COMPONENTS_OPERATIONLIST_HPP

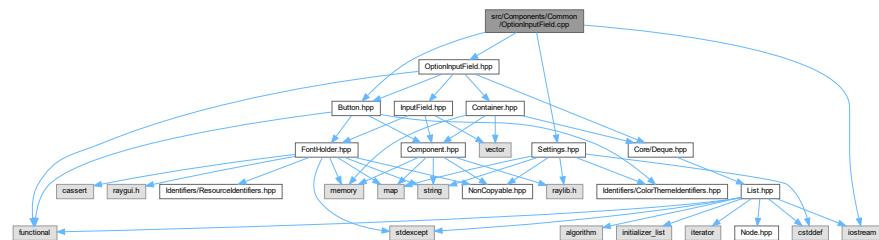
```

## 8.79 src/Components/Common/OptionInputField.cpp File Reference

```

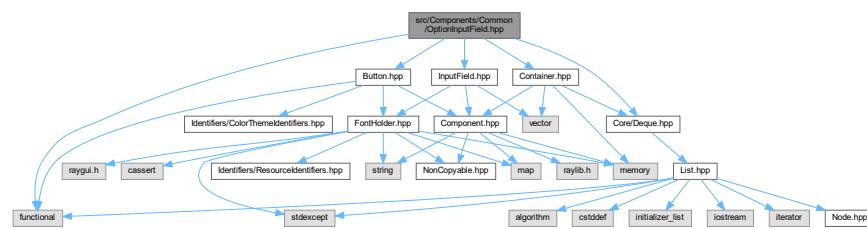
#include "OptionInputField.hpp"
#include <iostream>
#include "Button.hpp"
#include "Settings.hpp"
Include dependency graph for OptionInputField.cpp:

```

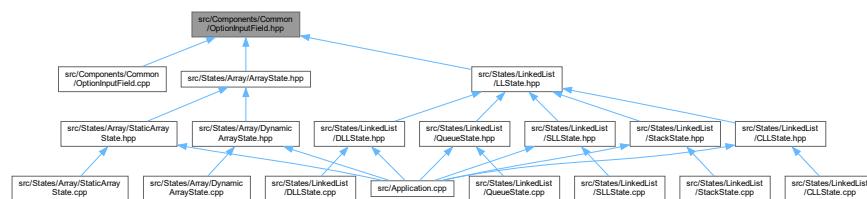


## 8.80 src/Components/Common/OptionInputField.hpp File Reference

```
#include <functional>
#include "Button.hpp"
#include "Container.hpp"
#include "Core/Deque.hpp"
#include "InputField.hpp"
Include dependency graph for OptionInputField.hpp:
```



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



### Classes

- class [GUIComponent::OptionInputField](#)  
*The option input field class that is used to represent an option input field in the GUI.*

### Namespaces

- namespace [GUIComponent](#)

## 8.81 OptionInputField.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef COMPONENTS_OPTIONINPUTFIELD_HPP
00002 #define COMPONENTS_OPTIONINPUTFIELD_HPP
00003
00004 #include <functional>
00005
00006 #include "Button.hpp"
00007 #include "Container.hpp"
00008 #include "Core/Deque.hpp"
00009 #include "InputField.hpp"
00010
00011 namespace GUIComponent {
```

```

00018     class OptionInputField : public GUI::Container {
00019     public:
00020         typedef std::shared_ptr< OptionInputField > Ptr;
00021
00022     public:
00023         OptionInputField(FontHolder *fonts);
00024         ~OptionInputField();
00025         void SetOption(
00026             std::string content, Core::Deque< InputField::Ptr > fields,
00027             std::function< void(std::map< std::string, std::string >) > action);
00028         void SetNoFieldOption(std::string content,
00029             std::function< void() > action);
00030
00031         void ToggleInputFields();
00032
00033         void DrawCurrent(Vector2 base = (Vector2){0, 0});
00034         void SetVisible(bool visible);
00035
00036     public:
00037         virtual Vector2 GetSize();
00038
00039     private:
00040         std::map< std::string, std::string > ExtractInput();
00041         void AddInputField(InputField::Ptr inputField);
00042         void AddSubmitField(
00043             std::function< void(std::map< std::string, std::string >) > action);
00044
00045     private:
00046         bool HasInputField();
00047         GUI::Container::Ptr mInputField;
00048         std::map< std::string, std::string > mInput;
00049
00050     private:
00051         FontHolder *fonts;
00052         bool hasInputField;
00053         // std::function< void(std::map< std::string, std::string >) > mAction;
00054     };
00055 }; // namespace GUIComponent
00056
00057 #endif // COMPONENTS_OPTIONINPUTFIELD_HPP

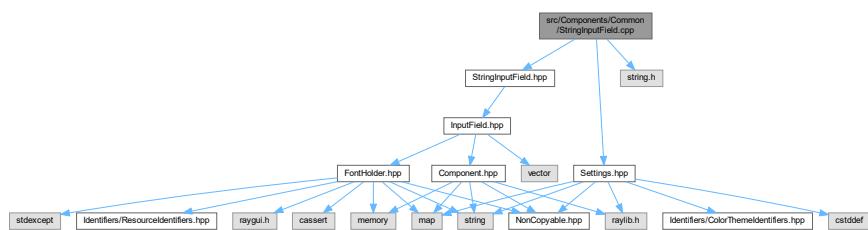
```

## 8.82 src/Components/Common/StringInputField.cpp File Reference

```

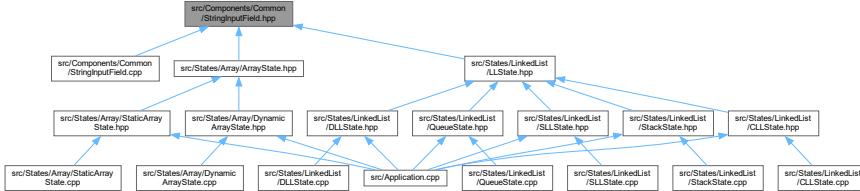
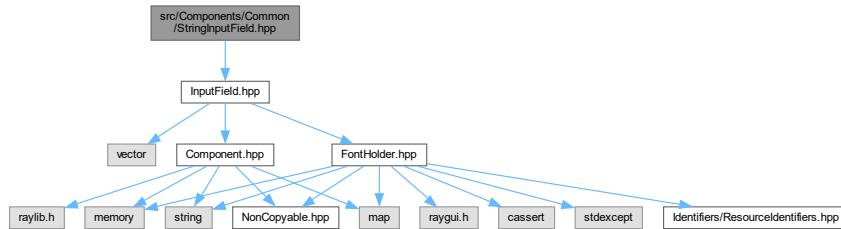
#include "StringInputField.hpp"
#include <string.h>
#include "Settings.hpp"
Include dependency graph for StringInputField.cpp:

```



## 8.83 src/Components/Common/StringInputField.hpp File Reference

```
#include "InputField.hpp"
Include dependency graph for StringInputField.hpp:
```



### Classes

- class [GUIComponent::StringInputField](#)  
*The string input field class that is used to represent a string input field in the GUI.*

### Namespaces

- namespace [GUIComponent](#)

## 8.84 StringInputField.hpp

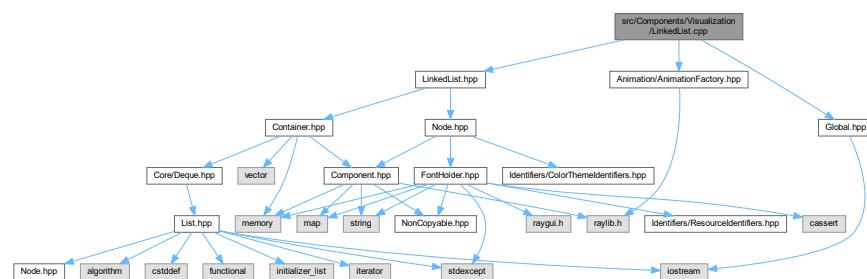
[Go to the documentation of this file.](#)

```

00001 #ifndef COMPONENTS_STRINGINPUTFIELD_HPP
00002 #define COMPONENTS_STRINGINPUTFIELD_HPP
00003
00004 #include "InputField.hpp"
00005
00006 namespace GUIComponent {
00013     class StringInputField : public InputField {
00014     public:
00015         typedef std::shared_ptr< StringInputField > Ptr;
00016         StringInputField(FontHolder* fonts);
00017         ~StringInputField();
00018         std::string ExtractValue();
00019         void DrawField(Vector2 base = (Vector2){0, 0});
00020         bool IsSelectable() const;
00021
00022     private:
00023         std::size_t mMaxLength;
00024         std::string content;
00025     };
00026 }; // namespace GUIComponent
00027
00028 #endif // COMPONENTS_STRINGINPUTFIELD_HPP
  
```

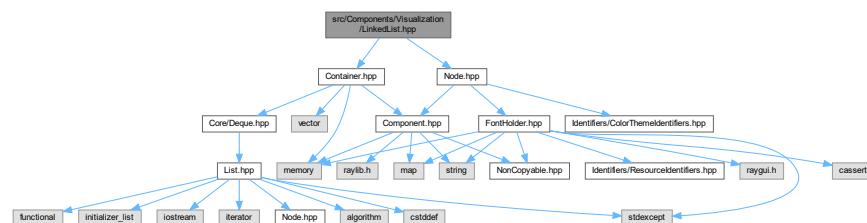
## 8.85 src/Components/Visualization/LinkedList.cpp File Reference

```
#include "LinkedList.hpp"
#include "Animation/AnimationFactory.hpp"
#include "Global.hpp"
Include dependency graph for LinkedList.cpp:
```

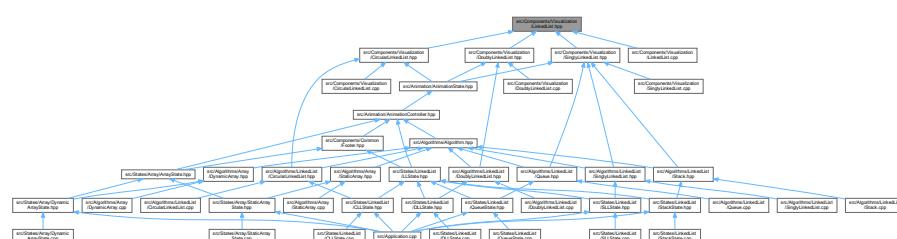


## 8.86 src/Components/Visualization/LinkedList.hpp File Reference

```
#include "Container.hpp"
#include "Node.hpp"
Include dependency graph for LinkedList.hpp:
```



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



## Classes

- class [GUIVisualizer::LinkedList](#)

*The base class for the linked list visualization. This class provides the basic functionality for the linked list visualization.*

## Namespaces

- namespace [GUIVisualizer](#)

## 8.87 LinkedList.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef COMPONENTS_VISUALIZATION_LINKEDLIST_HPP
00002 #define COMPONENTS_VISUALIZATION_LINKEDLIST_HPP
00003
00004 #include "Container.hpp"
00005 #include "Node.hpp"
00006
00007 namespace GUIVisualizer {
00008     class LinkedList : public GUI::Container {
00009     public:
00010         enum ArrowType {
00011             Default,
00012             Hidden,
00013             Active,
00014             Skip,
00015             ArrowTypeCount,
00016         };
00017
00018         enum Orientation {
00019             Horizontal,
00020             Vertical,
00021             OrientationCount,
00022         };
00023
00024     public:
00025         static constexpr float mNodeDistance = 20;
00026
00027     protected:
00028         FontHolder* fonts;
00029         std::vector< Node > list;
00030         bool mDisplayHeadAndTail;
00031
00032         Orientation mOrientation = Orientation::Horizontal;
00033
00034         Node::Shape mShape = Node::Shape::Circle;
00035
00036     public:
00037         void SetShape(Node::Shape shape);
00038
00039         Node::Shape GetShape() const;
00040
00041     public:
00042         LinkedList();
00043
00044         LinkedList(FontHolder* fonts);
00045
00046         ~LinkedList();
00047
00048         bool IsSelectable() const;
00049
00050         virtual void Draw(Vector2 base = (Vector2){0, 0}, float t = 1.0f,
00051                         bool init = false) = 0;
00052
00053         virtual std::size_t size() const;
00054
00055         virtual void SetShowHeadAndTail(bool show);
00056
00057         virtual void SetOrientation(Orientation orientation);
00058
00059     public:
00060         virtual std::vector< Node >& GetList();
00061         virtual Node GenerateNode(int value);
00062
00063         virtual void Import(std::vector< int > nodes);
00064
00065         virtual void InsertNode(std::size_t index, Node node,
00066                                bool rePosition = true);
00067
00068         virtual void DeleteNode(std::size_t index, bool rePosition = true);
00069
00070         virtual void Relayout();
00071
00072     protected:
00073
00074 }
```

```

00151     Vector2 GetNodeDefaultPosition(std::size_t index);
00152 };
00153 // namespace GUIVisualizer
00154
00155 #endif // COMPONENTS_VISUALIZATION_LINKEDLIST_HPP

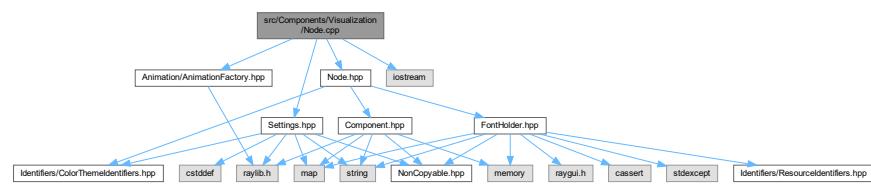
```

## 8.88 src/Components/Visualization/Node.cpp File Reference

```

#include "Node.hpp"
#include <iostream>
#include "Animation/AnimationFactory.hpp"
#include "Settings.hpp"
Include dependency graph for Node.cpp:

```

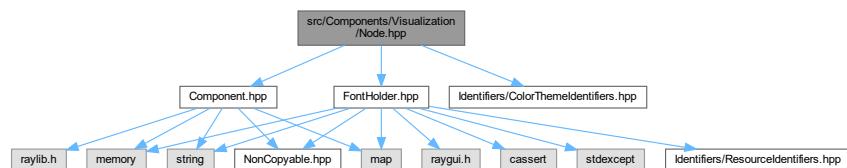


## 8.89 src/Components/Visualization/Node.hpp File Reference

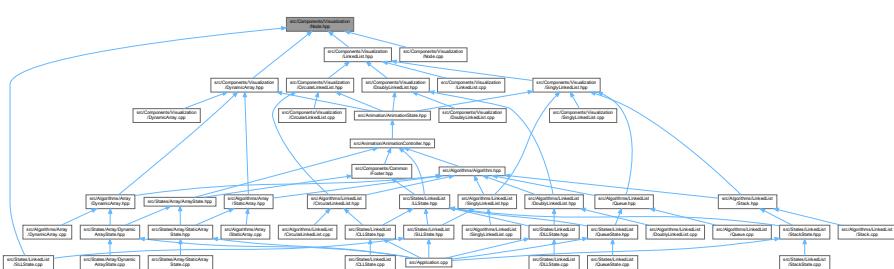
```

#include "Component.hpp"
#include "FontHolder.hpp"
#include "Identifiers/ColorThemeIdentifiers.hpp"
Include dependency graph for Node.hpp:

```



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



## Classes

- class [GUIVisualizer::Node](#)

*The node class that is used to represent a node in the visualization.*

## Namespaces

- namespace [GUIVisualizer](#)

## 8.90 Node.hpp

[Go to the documentation of this file.](#)

```

00001
00002 #ifndef COMPONENTS_VISUALIZATION_GUINODE_HPP
00003 #define COMPONENTS_VISUALIZATION_GUINODE_HPP
00004
00005 #include "Component.hpp"
00006 #include "FontHolder.hpp"
00007 #include "Identifiers/ColorThemeIdentifiers.hpp"
00008
00009 namespace GUIVisualizer {
00010     class Node : public GUI::Component {
00011     public:
00012         enum State {
00013             Default,
00014             Active,
00015             ActiveBlue,
00016             ActiveGreen,
00017             ActiveRed,
00018             Iterated,
00019             Hide,
00020             StateCount,
00021         };
00022
00023         enum Shape {
00024             Circle,
00025             Square,
00026             ShapeCount,
00027         };
00028
00029     private:
00030         Shape mShape = Shape::Circle;
00031
00032     public:
00033         void SetShape(Shape shape);
00034
00035         Shape GetShape() const;
00036
00037     public:
00038         Node(int value, FontHolder* fonts);
00039
00040         Node();
00041
00042         ~Node();
00043
00044         bool IsSelectable() const;
00045
00046         void Draw(Vector2 base = (Vector2){0, 0}, float t = 1.0f);
00047
00048         void SetActive(bool active);
00049
00050         bool IsActive();
00051
00052     public:
00053         void SetValue(int value);
00054
00055         int GetValue() const;
00056
00057         void SetLabel(std::string label);
00058
00059         void ClearLabel();
00060
00061     public:
00062         void AnimationOnNode(bool animate);
00063
00064 }
```

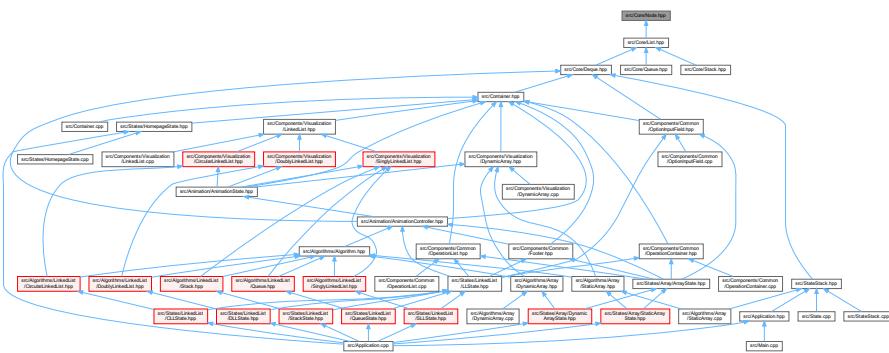
```

00131     void SetRadius(float radius);
00132
00133     private:
00134         void DrawLabel(Vector2 base = (Vector2){0, 0});
00135
00136         void DrawNode(Vector2 base = (Vector2){0, 0}, float t = 1.0f);
00137
00138     public:
00139         void SetValueFontSize(int fontSize);
00140         void SetLabelFontSize(int fontSize);
00141
00142         void SetNodeState(State state);
00143
00144         State GetNodeState() const;
00145
00146     private:
00147         Color GetOutlineColor(float t = 1.0f);
00148         Color GetBackgroundColor(float t = 1.0f);
00149         Color GetTextColor(float t = 1.0f);
00150         void AddColor();
00151
00152     public:
00153         void SetReachable(bool reachable);
00154         bool GetReachable() const;
00155
00156     private:
00157         std::map< State, std::pair< ColorTheme::ID, ColorTheme::ID > >
00158             mOutlineColor;
00159         std::map< State, std::pair< ColorTheme::ID, ColorTheme::ID > >
00160             mBackgroundColor;
00161         std::map< State, std::pair< ColorTheme::ID, ColorTheme::ID > >
00162             mTextColor;
00163
00164     private:
00165         int mValue;
00166         float mRadius;
00167         float valueFontSize;
00168         float labelFontSize;
00169
00170         std::string mLabel;
00171         State mNodeState;
00172         bool mReachable;
00173
00174     private:
00175         bool animateNode;
00176         bool mActive;
00177         FontHolder* fonts;
00178         Color mDefaultColor;
00179         Color mActiveColor;
00180         Color mBorderColor;
00181     };
00182 };
00183 // namespace GUIVisualizer
00184
00185 #endif // COMPONENTS_VISUALIZATION_GUINODE_HPP

```

## 8.91 src/Core/Node.hpp File Reference

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



## Classes

- class [Core::Node< T >](#)

*The node class that is used to store the value of the node, and the pointers to the previous and the next node (similar to Doubly Linked List node).*

## Namespaces

- namespace [Core](#)

## 8.92 Node.hpp

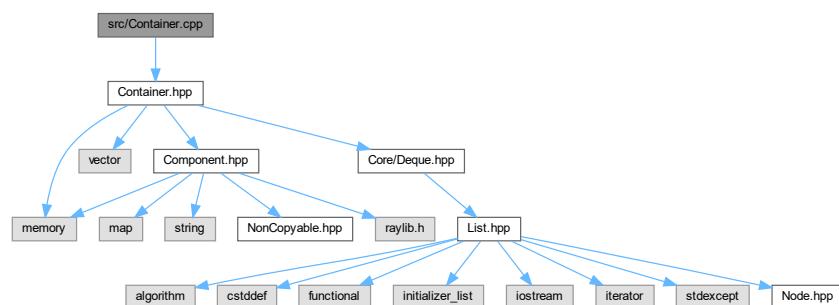
[Go to the documentation of this file.](#)

```
00001 #ifndef CORE_NODE_HPP
00002 #define CORE_NODE_HPP
00003
00004 namespace Core {
00011     template< typename T >
00012     class Node {
00013     public:
00017         T mValue;
00018
00022         Node< T *>* mPrev;
00023
00027         Node< T *>* mNext;
00028
00029     public:
00033         Node() : mValue{T()}, mPrev{nullptr}, mNext{nullptr} {}
00034
00039         Node(const T& value) : mValue{value}, mPrev{nullptr}, mNext{nullptr} {}
00040
00045         Node(const Node< T >& node)
00046             : mValue{node.mValue}, mPrev{node.mPrev}, mNext{node.mNext} {}
00047
00054         Node(const T& value, Node< T *>* const& prev, Node< T *>* const& next)
00055             : mValue{value}, mPrev{prev}, mNext{next} {}
00056     };
00057 } // namespace Core
00058
00059 #endif // CORE_NODE_HPP
```

## 8.93 src/Container.cpp File Reference

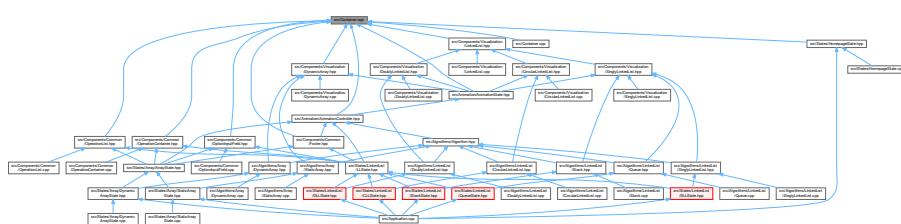
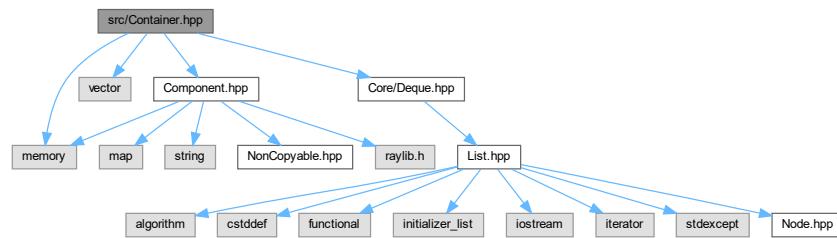
#include "Container.hpp"

Include dependency graph for Container.cpp:



## 8.94 src/Container.hpp File Reference

```
#include <memory>
#include <vector>
#include "Component.hpp"
#include "Core/Deque.hpp"
Include dependency graph for Container.hpp:
```



## Classes

- class [GUI::Container](#)  
*The base container class that is used to represent a [GUI](#) container in the [GUI](#).*

## Namespaces

- namespace [GUI](#)

## 8.95 Container.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef CONTAINER_HPP
00002 #define CONTAINER_HPP
00003
00004 #include <memory>
00005 #include <vector>
00006
00007 #include "Component.hpp"
00008 #include "Core/Deque.hpp"
00009
00010 namespace GUI {
00015     class Container : public Component {
```

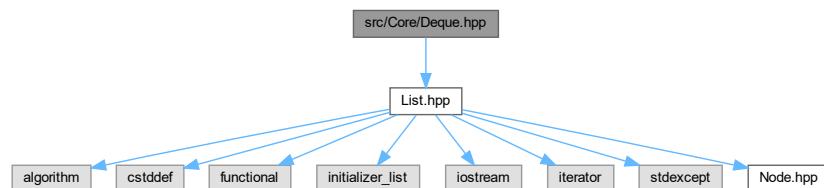
```

00016     public:
00023         typedef std::shared_ptr< Container > Ptr;
00024
00025     public:
00029         Container();
00030
00036         void pack(Component::Ptr component);
00037
00043         void UnpackAll();
00044
00050         virtual void Draw(Vector2 base = (Vector2){0, 0});
00051
00058         virtual bool isSelectable() const;
00059
00066         Core::Deque< Component::Ptr > GetChildren();
00067
00068         virtual void DrawCurrent(Vector2 base);
00069
00075         virtual Vector2 GetSize();
00076
00077     protected:
00078         // std::vector< Component::Ptr > mChildren;
00079         Core::Deque< Component::Ptr > mChildren;
00080     };
00081 }; // namespace GUI
00082
00083 #endif // CONTAINER_HPP

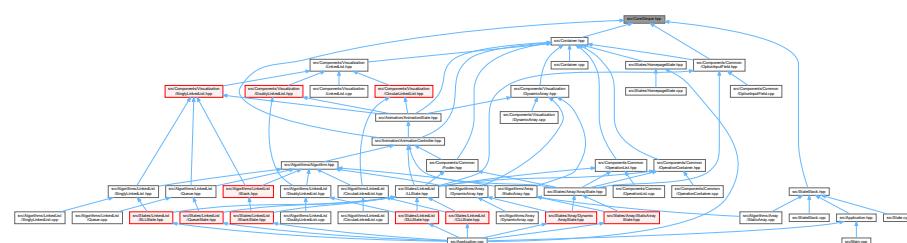
```

## 8.96 src/Core/Deque.hpp File Reference

#include "List.hpp"  
Include dependency graph for Deque.hpp:



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



## Classes

- class **Core::Deque< T >**

*The deque container.*

## Namespaces

- namespace **Core**

## 8.97 Deque.hpp

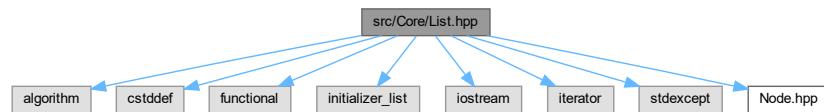
[Go to the documentation of this file.](#)

```
00001 #ifndef CORE_DEQUE_HPP
00002 #define CORE_DEQUE_HPP
00003
00004 #include "List.hpp"
00005
00006 namespace Core {
00011     template< typename T >
00012     class Deque : public List< T > {
00013     private:
00014         using List = Core::List< T >;
00015
00016     public:
00017         using List::assign;
00018         using List::List;
00019         using List::operator=;
00020         using List::swap;
00021
00022     public:
00023         using List::clear;
00024         using List::empty;
00025         using List::size;
00026
00027     public:
00028         using List::back;
00029         using List::front;
00030
00031     public:
00032         using List::push_back;
00033         using List::push_front;
00034
00035     public:
00036         using List::pop_back;
00037         using List::pop_front;
00038
00039     public:
00040         using List::begin;
00041         using List::end;
00042
00043     public:
00044         using List::resize;
00045
00046     public:
00047         using List::remove;
00048         using List::reverse;
00049         using List::unique;
00050
00051     public:
00052         using List::at;
00053         using List::operator[];
00054     };
00055 }; // namespace Core
00056
00057 #endif // CORE_DEQUE_HPP
```

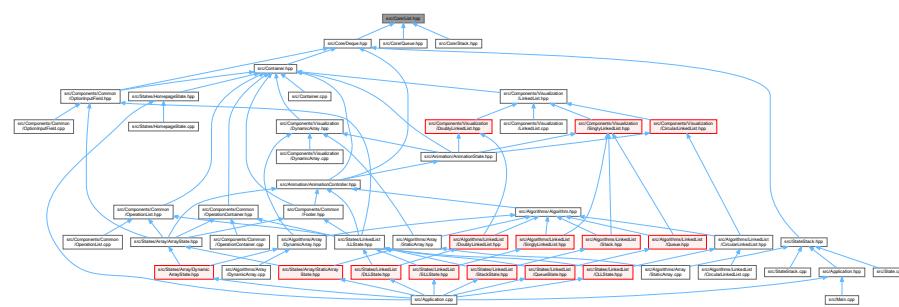
## 8.98 src/Core/List.hpp File Reference

```
#include <algorithm>
#include <cstddef>
#include <functional>
#include <initializer_list>
#include <iostream>
#include <iterator>
#include <stdexcept>
```

```
#include "Node.hpp"
Include dependency graph for List.hpp:
```



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



## Classes

- class [Core::List< T >](#)  
*The base container for implementing other data structures.*
- class [Core::List< T >::iterator](#)  
*The list iterator class.*
- class [Core::List< T >::const\\_iterator](#)  
*The list const\_iterator class.*

## Namespaces

- namespace [Core](#)

## 8.99 List.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef CORE_LIST_HPP
00002 #define CORE_LIST_HPP
00003
00004 // #define _SILENCE_ALL_CXX17_DEPRECATED_WARNINGS
00005 // #define _SILENCE_CXX17_ITERATOR_BASE_CLASS_DEPRECATED_WARNING
00006
00007 #include <algorithm>
00008 #include <cstddef>
00009 #include <functional>
00010 #include <initializer_list>
00011 #include <iostream>
00012 #include <iterator>
00013 #include <stdexcept>
00014
```

```

00015 #include "Node.hpp"
00016
00017 namespace Core {
00022     template< typename T >
00023     class List {
00024     public:
00025         class iterator;
00026         class const_iterator;
00027
00028     private:
00029         iterator mBegin;
00030
00031         iterator mEnd;
00032
00033         std::size_t mSize;
00034
00035         void reset() {
00036             this->mBegin = this->mEnd;
00037             this->mEnd.ptr->mPrev = nullptr;
00038             this->mSize = 0;
00039         }
00040
00041         void insert_previous(const iterator& it, const iterator& it_prev) {
00042             Node< T *>* node = it.ptr;
00043             Node< T *>* node_prev = it_prev.ptr;
00044
00045             node_prev->mNext = node;
00046             node_prev->mPrev = node->mPrev;
00047             // node->mPrev = node_prev;
00048             node_prev->mNext->mPrev = node_prev;
00049             if (node_prev->mPrev != nullptr) {
00050                 node_prev->mPrev->mNext = node_prev;
00051             }
00052
00053             if (it == this->begin()) {
00054                 this->mBegin = node_prev;
00055             }
00056
00057             ++this->mSize;
00058         }
00059
00060         /* @brief Move nodes from a list of range ['first', 'last') to before
00061          * 'it'
00062          * @param it The iterator to insert the nodes before
00063          * @return iterator pointing to the first inserted element, or
00064          * 'it' if 'first' == 'last'.
00065          * @exception: undefined behaviour: null pointer reference
00066          */
00067         iterator move_previous(const iterator& it, const iterator& first,
00068                               const iterator& last) {
00069             mSize += std::distance(first, last);
00070
00071             iterator prev = it;
00072             prev--;
00073
00074             if (first != last) {
00075                 Node< T *>* node = it.ptr;
00076                 Node< T *>* node_first = first.ptr;
00077                 Node< T *>* node_last = last.ptr->mPrev;
00078
00079                 node_first->mPrev = node->mPrev;
00080                 if (node->mPrev != nullptr) {
00081                     node->mPrev->mNext = node_first;
00082                 }
00083
00084                 node_last->mNext = node;
00085                 node->mPrev = node_last;
00086
00087                 if (it == this->begin()) {
00088                     this->mBegin = node_first;
00089                 }
00090             }
00091
00092             return prev == nullptr ? this->begin() : prev++;
00093         }
00094
00095         iterator get_iterator(std::size_t index) {
00096             if (!(index >= 0 && index < this->size())) {
00097                 throw std::out_of_range("Index out of range");
00098             }
00099
00100             iterator it;
00101             if (index < this->size() / 2) {
00102                 it = this->begin();
00103                 for (std::size_t i = 0; i < index; ++i) ++it;
00104             } else {
00105                 it = this->end();
00106             }
00107         }
00108
00109         iterator begin() { return mBegin; }
00110         iterator end() { return mEnd; }
00111
00112         std::size_t size() const { return mSize; }
00113
00114         void clear() {
00115             mBegin = mEnd;
00116             mSize = 0;
00117         }
00118
00119         void swap(List &other) {
00120             std::swap(mBegin, other.mBegin);
00121             std::swap(mEnd, other.mEnd);
00122             std::swap(mSize, other.mSize);
00123         }
00124
00125     };
00126
00127     template< typename T >
00128     class const_iterator {
00129     public:
00130         const Node< T *>* ptr;
00131
00132         const_iterator(const Node< T *>* p) : ptr(p) {}
00133
00134         const_iterator(const const_iterator &other) : ptr(other.ptr) {}
00135
00136         const_iterator &operator=(const const_iterator &other) {
00137             ptr = other.ptr;
00138             return *this;
00139         }
00140
00141         const_iterator operator++(int) {
00142             const_iterator result(*this);
00143             ++ptr;
00144             return result;
00145         }
00146
00147         const_iterator &operator--(int) {
00148             const_iterator result(*this);
00149             --ptr;
00150             return result;
00151         }
00152
00153         const_iterator operator+(std::size_t n) const {
00154             const_iterator result(*this);
00155             result += n;
00156             return result;
00157         }
00158
00159         const_iterator &operator+=(std::size_t n) {
00160             for (std::size_t i = 0; i < n; ++i) ++ptr;
00161             return *this;
00162         }
00163
00164         const_iterator operator-(std::size_t n) const {
00165             const_iterator result(*this);
00166             result -= n;
00167             return result;
00168         }
00169
00170         const_iterator &operator-=(std::size_t n) {
00171             for (std::size_t i = 0; i < n; ++i) --ptr;
00172             return *this;
00173         }
00174
00175         bool operator==(const const_iterator &other) const {
00176             return ptr == other.ptr;
00177         }
00178
00179         bool operator!=(const const_iterator &other) const {
00180             return ptr != other.ptr;
00181         }
00182
00183         void operator*() const {
00184             if (ptr == nullptr)
00185                 throw std::invalid_argument("Null pointer dereference");
00186             *ptr;
00187         }
00188
00189         const_iterator &operator*() {
00190             if (ptr == nullptr)
00191                 throw std::invalid_argument("Null pointer dereference");
00192             *ptr;
00193             return *this;
00194         }
00195
00196         const_iterator &operator[](std::size_t index) {
00197             if (index >= mSize)
00198                 throw std::out_of_range("Index out of range");
00199             return *this;
00200         }
00201
00202         const Node< T *>* operator->() const {
00203             if (ptr == nullptr)
00204                 throw std::invalid_argument("Null pointer dereference");
00205             return ptr;
00206         }
00207
00208         const Node< T *>* operator->() {
00209             if (ptr == nullptr)
00210                 throw std::invalid_argument("Null pointer dereference");
00211             return ptr;
00212             return *this;
00213         }
00214
00215         friend std::ostream &operator<<(std::ostream &os, const const_iterator &it) {
00216             os << *it.ptr;
00217             return os;
00218         }
00219
00220     };
00221
00222     template< typename T >
00223     class iterator {
00224     public:
00225         Node< T *>* ptr;
00226
00227         iterator(Node< T *>* p) : ptr(p) {}
00228
00229         iterator(const const_iterator &other) : ptr(other.ptr) {}
00230
00231         iterator &operator=(const const_iterator &other) {
00232             ptr = other.ptr;
00233             return *this;
00234         }
00235
00236         iterator &operator++() {
00237             ++ptr;
00238             return *this;
00239         }
00240
00241         iterator &operator--() {
00242             --ptr;
00243             return *this;
00244         }
00245
00246         iterator &operator+=(std::size_t n) {
00247             for (std::size_t i = 0; i < n; ++i) ++ptr;
00248             return *this;
00249         }
00250
00251         iterator &operator-=(std::size_t n) {
00252             for (std::size_t i = 0; i < n; ++i) --ptr;
00253             return *this;
00254         }
00255
00256         friend std::ostream &operator<<(std::ostream &os, const iterator &it) {
00257             os << *it.ptr;
00258             return os;
00259         }
00260
00261     };
00262
00263     template< typename T >
00264     class const_node {
00265     public:
00266         const Node< T *>* ptr;
00267
00268         const_node(const Node< T *>* p) : ptr(p) {}
00269
00270         const_node(const const_node &other) : ptr(other.ptr) {}
00271
00272         const_node &operator=(const const_node &other) {
00273             ptr = other.ptr;
00274             return *this;
00275         }
00276
00277         friend std::ostream &operator<<(std::ostream &os, const const_node &it) {
00278             os << *it.ptr;
00279             return os;
00280         }
00281
00282     };
00283
00284     template< typename T >
00285     class node {
00286     public:
00287         Node< T *>* ptr;
00288
00289         node(Node< T *>* p) : ptr(p) {}
00290
00291         node(const const_node &other) : ptr(other.ptr) {}
00292
00293         node &operator=(const const_node &other) {
00294             ptr = other.ptr;
00295             return *this;
00296         }
00297
00298         friend std::ostream &operator<<(std::ostream &os, const node &it) {
00299             os << *it.ptr;
00300             return os;
00301         }
00302
00303     };
00304
00305     template< typename T >
00306     class const_node {
00307     public:
00308         const Node< T *>* ptr;
00309
00310         const_node(const Node< T *>* p) : ptr(p) {}
00311
00312         const_node(const const_node &other) : ptr(other.ptr) {}
00313
00314         const_node &operator=(const const_node &other) {
00315             ptr = other.ptr;
00316             return *this;
00317         }
00318
00319         friend std::ostream &operator<<(std::ostream &os, const const_node &it) {
00320             os << *it.ptr;
00321             return os;
00322         }
00323
00324     };
00325
00326     template< typename T >
00327     class node {
00328     public:
00329         Node< T *>* ptr;
00330
00331         node(Node< T *>* p) : ptr(p) {}
00332
00333         node(const const_node &other) : ptr(other.ptr) {}
00334
00335         node &operator=(const const_node &other) {
00336             ptr = other.ptr;
00337             return *this;
00338         }
00339
00340         friend std::ostream &operator<<(std::ostream &os, const node &it) {
00341             os << *it.ptr;
00342             return os;
00343         }
00344
00345     };
00346
00347     template< typename T >
00348     class const_iterator {
00349     public:
00350         const Node< T *>* ptr;
00351
00352         const_iterator(const Node< T *>* p) : ptr(p) {}
00353
00354         const_iterator(const const_iterator &other) : ptr(other.ptr) {}
00355
00356         const_iterator &operator=(const const_iterator &other) {
00357             ptr = other.ptr;
00358             return *this;
00359         }
00360
00361         const_iterator &operator++() {
00362             ++ptr;
00363             return *this;
00364         }
00365
00366         const_iterator &operator--() {
00367             --ptr;
00368             return *this;
00369         }
00370
00371         const_iterator &operator+=(std::size_t n) {
00372             for (std::size_t i = 0; i < n; ++i) ++ptr;
00373             return *this;
00374         }
00375
00376         const_iterator &operator-=(std::size_t n) {
00377             for (std::size_t i = 0; i < n; ++i) --ptr;
00378             return *this;
00379         }
00380
00381         friend std::ostream &operator<<(std::ostream &os, const const_iterator &it) {
00382             os << *it.ptr;
00383             return os;
00384         }
00385
00386     };
00387
00388     template< typename T >
00389     class iterator {
00390     public:
00391         Node< T *>* ptr;
00392
00393         iterator(Node< T *>* p) : ptr(p) {}
00394
00395         iterator(const const_iterator &other) : ptr(other.ptr) {}
00396
00397         iterator &operator=(const const_iterator &other) {
00398             ptr = other.ptr;
00399             return *this;
00400         }
00401
00402         iterator &operator++() {
00403             ++ptr;
00404             return *this;
00405         }
00406
00407         iterator &operator--() {
00408             --ptr;
00409             return *this;
00410         }
00411
00412         iterator &operator+=(std::size_t n) {
00413             for (std::size_t i = 0; i < n; ++i) ++ptr;
00414             return *this;
00415         }
00416
00417         iterator &operator-=(std::size_t n) {
00418             for (std::size_t i = 0; i < n; ++i) --ptr;
00419             return *this;
00420         }
00421
00422         friend std::ostream &operator<<(std::ostream &os, const iterator &it) {
00423             os << *it.ptr;
00424             return os;
00425         }
00426
00427     };
00428
00429     template< typename T >
00430     class const_node {
00431     public:
00432         const Node< T *>* ptr;
00433
00434         const_node(const Node< T *>* p) : ptr(p) {}
00435
00436         const_node(const const_node &other) : ptr(other.ptr) {}
00437
00438         const_node &operator=(const const_node &other) {
00439             ptr = other.ptr;
00440             return *this;
00441         }
00442
00443         friend std::ostream &operator<<(std::ostream &os, const const_node &it) {
00444             os << *it.ptr;
00445             return os;
00446         }
00447
00448     };
00449
00450     template< typename T >
00451     class node {
00452     public:
00453         Node< T *>* ptr;
00454
00455         node(Node< T *>* p) : ptr(p) {}
00456
00457         node(const const_node &other) : ptr(other.ptr) {}
00458
00459         node &operator=(const const_node &other) {
00460             ptr = other.ptr;
00461             return *this;
00462         }
00463
00464         friend std::ostream &operator<<(std::ostream &os, const node &it) {
00465             os << *it.ptr;
00466             return os;
00467         }
00468
00469     };
00470
00471     template< typename T >
00472     class const_iterator {
00473     public:
00474         const Node< T *>* ptr;
00475
00476         const_iterator(const Node< T *>* p) : ptr(p) {}
00477
00478         const_iterator(const const_iterator &other) : ptr(other.ptr) {}
00479
00480         const_iterator &operator=(const const_iterator &other) {
00481             ptr = other.ptr;
00482             return *this;
00483         }
00484
00485         const_iterator &operator++() {
00486             ++ptr;
00487             return *this;
00488         }
00489
00490         const_iterator &operator--() {
00491             --ptr;
00492             return *this;
00493         }
00494
00495         const_iterator &operator+=(std::size_t n) {
00496             for (std::size_t i = 0; i < n; ++i) ++ptr;
00497             return *this;
00498         }
00499
00500         const_iterator &operator-=(std::size_t n) {
00501             for (std::size_t i = 0; i < n; ++i) --ptr;
00502             return *this;
00503         }
00504
00505         friend std::ostream &operator<<(std::ostream &os, const const_iterator &it) {
00506             os << *it.ptr;
00507             return os;
00508         }
00509
00510     };
00511
00512     template< typename T >
00513     class iterator {
00514     public:
00515         Node< T *>* ptr;
00516
00517         iterator(Node< T *>* p) : ptr(p) {}
00518
00519         iterator(const const_iterator &other) : ptr(other.ptr) {}
00520
00521         iterator &operator=(const const_iterator &other) {
00522             ptr = other.ptr;
00523             return *this;
00524         }
00525
00526         iterator &operator++() {
00527             ++ptr;
00528             return *this;
00529         }
00530
00531         iterator &operator--() {
00532             --ptr;
00533             return *this;
00534         }
00535
00536         iterator &operator+=(std::size_t n) {
00537             for (std::size_t i = 0; i < n; ++i) ++ptr;
00538             return *this;
00539         }
00540
00541         iterator &operator-=(std::size_t n) {
00542             for (std::size_t i = 0; i < n; ++i) --ptr;
00543             return *this;
00544         }
00545
00546         friend std::ostream &operator<<(std::ostream &os, const iterator &it) {
00547             os << *it.ptr;
00548             return os;
00549         }
00550
00551     };
00552
00553     template< typename T >
00554     class const_node {
00555     public:
00556         const Node< T *>* ptr;
00557
00558         const_node(const Node< T *>* p) : ptr(p) {}
00559
00560         const_node(const const_node &other) : ptr(other.ptr) {}
00561
00562         const_node &operator=(const const_node &other) {
00563             ptr = other.ptr;
00564             return *this;
00565         }
00566
00567         friend std::ostream &operator<<(std::ostream &os, const const_node &it) {
00568             os << *it.ptr;
00569             return os;
00570         }
00571
00572     };
00573
00574     template< typename T >
00575     class node {
00576     public:
00577         Node< T *>* ptr;
00578
00579         node(Node< T *>* p) : ptr(p) {}
00580
00581         node(const const_node &other) : ptr(other.ptr) {}
00582
00583         node &operator=(const const_node &other) {
00584             ptr = other.ptr;
00585             return *this;
00586         }
00587
00588         friend std::ostream &operator<<(std::ostream &os, const node &it) {
00589             os << *it.ptr;
00590             return os;
00591         }
00592
00593     };
00594
00595     template< typename T >
00596     class const_iterator {
00597     public:
00598         const Node< T *>* ptr;
00599
00600         const_iterator(const Node< T *>* p) : ptr(p) {}
00601
00602         const_iterator(const const_iterator &other) : ptr(other.ptr) {}
00603
00604         const_iterator &operator=(const const_iterator &other) {
00605             ptr = other.ptr;
00606             return *this;
00607         }
00608
00609         const_iterator &operator++() {
00610             ++ptr;
00611             return *this;
00612         }
00613
00614         const_iterator &operator--() {
00615             --ptr;
00616             return *this;
00617         }
00618
00619         const_iterator &operator+=(std::size_t n) {
00620             for (std::size_t i = 0; i < n; ++i) ++ptr;
00621             return *this;
00622         }
00623
00624         const_iterator &operator-=(std::size_t n) {
00625             for (std::size_t i = 0; i < n; ++i) --ptr;
00626             return *this;
00627         }
00628
00629         friend std::ostream &operator<<(std::ostream &os, const const_iterator &it) {
00630             os << *it.ptr;
00631             return os;
00632         }
00633
00634     };
00635
00636     template< typename T >
00637     class iterator {
00638     public:
00639         Node< T *>* ptr;
00640
00641         iterator(Node< T *>* p) : ptr(p) {}
00642
00643         iterator(const const_iterator &other) : ptr(other.ptr) {}
00644
00645         iterator &operator=(const const_iterator &other) {
00646             ptr = other.ptr;
00647             return *this;
00648         }
00649
00650         iterator &operator++() {
00651             ++ptr;
00652             return *this;
00653         }
00654
00655         iterator &operator--() {
00656             --ptr;
00657             return *this;
00658         }
00659
00660         iterator &operator+=(std::size_t n) {
00661             for (std::size_t i = 0; i < n; ++i) ++ptr;
00662             return *this;
00663         }
00664
00665         iterator &operator-=(std::size_t n) {
00666             for (std::size_t i = 0; i < n; ++i) --ptr;
00667             return *this;
00668         }
00669
00670         friend std::ostream &operator<<(std::ostream &os, const iterator &it) {
00671             os << *it.ptr;
00672             return os;
00673         }
00674
00675     };
00676
00677     template< typename T >
00678     class const_node {
00679     public:
00680         const Node< T *>* ptr;
00681
00682         const_node(const Node< T *>* p) : ptr(p) {}
00683
00684         const_node(const const_node &other) : ptr(other.ptr) {}
00685
00686         const_node &operator=(const const_node &other) {
00687             ptr = other.ptr;
00688             return *this;
00689         }
00690
00691         friend std::ostream &operator<<(std::ostream &os, const const_node &it) {
00692             os << *it.ptr;
00693             return os;
00694         }
00695
00696     };
00697
00698     template< typename T >
00699     class node {
00700     public:
00701         Node< T *>* ptr;
00702
00703         node(Node< T *>* p) : ptr(p) {}
00704
00705         node(const const_node &other) : ptr(other.ptr) {}
00706
00707         node &operator=(const const_node &other) {
00708             ptr = other.ptr;
00709             return *this;
00710         }
00711
00712         friend std::ostream &operator<<(std::ostream &os, const node &it) {
00713             os << *it.ptr;
00714             return os;
00715         }
00716
00717     };
00718
00719     template< typename T >
00720     class const_iterator {
00721     public:
00722         const Node< T *>* ptr;
00723
00724         const_iterator(const Node< T *>* p) : ptr(p) {}
00725
00726         const_iterator(const const_iterator &other) : ptr(other.ptr) {}
00727
00728         const_iterator &operator=(const const_iterator &other) {
00729             ptr = other.ptr;
00730             return *this;
00731         }
00732
00733         const_iterator &operator++() {
00734             ++ptr;
00735             return *this;
00736         }
00737
00738         const_iterator &operator--() {
00739             --ptr;
00740             return *this;
00741         }
00742
00743         const_iterator &operator+=(std::size_t n) {
00744             for (std::size_t i = 0; i < n; ++i) ++ptr;
00745             return *this;
00746         }
00747
00748         const_iterator &operator-=(std::size_t n) {
00749             for (std::size_t i = 0; i < n; ++i) --ptr;
00750             return *this;
00751         }
00752
00753         friend std::ostream &operator<<(std::ostream &os, const const_iterator &it) {
00754             os << *it.ptr;
00755             return os;
00756         }
00757
00758     };
00759
00760     template< typename T >
00761     class iterator {
00762     public:
00763         Node< T *>* ptr;
00764
00765         iterator(Node< T *>* p) : ptr(p) {}
00766
00767         iterator(const const_iterator &other) : ptr(other.ptr) {}
00768
00769         iterator &operator=(const const_iterator &other) {
00770             ptr = other.ptr;
00771             return *this;
00772         }
00773
00774         iterator &operator++() {
00775             ++ptr;
00776             return *this;
00777         }
00778
00779         iterator &operator--() {
00780             --ptr;
00781             return *this;
00782         }
00783
00784         iterator &operator+=(std::size_t n) {
00785             for (std::size_t i = 0; i < n; ++i) ++ptr;
00786             return *this;
00787         }
00788
00789         iterator &operator-=(std::size_t n) {
00790             for (std::size_t i = 0; i < n; ++i) --ptr;
00791             return *this;
00792         }
00793
00794         friend std::ostream &operator<<(std::ostream &os, const iterator &it) {
00795             os << *it.ptr;
00796             return os;
00797         }
00798
00799     };
00800
00801     template< typename T >
00802     class const_node {
00803     public:
00804         const Node< T *>* ptr;
00805
00806         const_node(const Node< T *>* p) : ptr(p) {}
00807
00808         const_node(const const_node &other) : ptr(other.ptr) {}
00809
00810         const_node &operator=(const const_node &other) {
00811             ptr = other.ptr;
00812             return *this;
00813         }
00814
00815         friend std::ostream &operator<<(std::ostream &os, const const_node &it) {
00816             os << *it.ptr;
00817             return os;
00818         }
00819
00820     };
00821
00822     template< typename T >
00823     class node {
00824     public:
00825         Node< T *>* ptr;
00826
00827         node(Node< T *>* p) : ptr(p) {}
00828
00829         node(const const_node &other) : ptr(other.ptr) {}
00830
00831         node &operator=(const const_node &other) {
00832             ptr = other.ptr;
00833             return *this;
00834         }
00835
00836         friend std::ostream &operator<<(std::ostream &os, const node &it) {
00837             os << *it.ptr;
00838             return os;
00839         }
00840
00841     };
00842
00843     template< typename T >
00844     class const_iterator {
00845     public:
00846         const Node< T *>* ptr;
00847
00848         const_iterator(const Node< T *>* p) : ptr(p) {}
00849
00850         const_iterator(const const_iterator &other) : ptr(other.ptr) {}
00851
00852         const_iterator &operator=(const const_iterator &other) {
00853             ptr = other.ptr;
00854             return *this;
00855         }
00856
00857         const_iterator &operator++() {
00858             ++ptr;
00859             return *this;
00860         }
00861
00862         const_iterator &operator--() {
00863             --ptr;
00864             return *this;
00865         }
00866
00867         const_iterator &operator+=(std::size_t n) {
00868             for (std::size_t i = 0; i < n; ++i) ++ptr;
00869             return *this;
00870         }
00871
00872         const_iterator &operator-=(std::size_t n) {
00873             for (std::size_t i = 0; i < n; ++i) --ptr;
00874             return *this;
00875         }
00876
00877         friend std::ostream &operator<<(std::ostream &os, const const_iterator &it) {
00878             os << *it.ptr;
00879             return os;
00880         }
00881
00882     };
00883
00884     template< typename T >
00885     class iterator {
00886     public:
00887         Node< T *>* ptr;
00888
00889         iterator(Node< T *>* p) : ptr(p) {}
00890
00891         iterator(const const_iterator &other) : ptr(other.ptr) {}
00892
00893         iterator &operator=(const const_iterator &other) {
00894             ptr = other.ptr;
00895             return *this;
00896         }
00897
00898         iterator &operator++() {
00899             ++ptr;
00900             return *this;
00901         }
00902
00903         iterator &operator--() {
00904             --ptr;
00905             return *this;
00906         }
00907
00908         iterator &operator+=(std::size_t n) {
00909             for (std::size_t i = 0; i < n; ++i) ++ptr;
00910             return *this;
00911         }
00912
00913         iterator &operator-=(std::size_t n) {
00914             for (std::size_t i = 0; i < n; ++i) --ptr;
00915             return *this;
00916         }
00917
00918         friend std::ostream &operator<<(std::ostream &os, const iterator &it) {
00919             os << *it.ptr;
00920             return os;
00921         }
00922
00923     };
00924
00925     template< typename T >
00926     class const_node {
00927     public:
00928         const Node< T *>* ptr;
00929
00930         const_node(const Node< T *>* p) : ptr(p) {}
00931
00932         const_node(const const_node &other) : ptr(other.ptr) {}
00933
00934         const_node &operator=(const const_node &other) {
00935             ptr = other.ptr;
00936             return *this;
00937         }
00938
00939         friend std::ostream &operator<<(std::ostream &os, const const_node &it) {
00940             os << *it.ptr;
00941             return os;
00942         }
00943
00944     };
00945
00946     template< typename T &
```

```

00126             for (std::size_t i = this->size() - 1; i >= index; --i) --it;
00127         }
00128         return it;
00129     }
00130
00131     public:
00132     List() : mSize(0) { this->mBegin = this->mEnd = new Node< T >(); }
00133
00134     List(std::initializer_list< T > list) : mSize(0) {
00135         this->mBegin = this->mEnd = new Node< T >();
00136         for (const auto& element : list) {
00137             this->push_back(element);
00138         }
00139     }
00140
00141     ~List() {
00142         this->clear();
00143         delete mEnd.ptr;
00144     }
00145
00146     List(const List< T >& list) : mSize(0) {
00147         this->mBegin = this->mEnd = new Node< T >();
00148         for (const auto& element : list) {
00149             this->push_back(element);
00150         }
00151     }
00152
00153     List(List< T >&& list) : mSize(0) {
00154         this->mBegin = this->mEnd = new Node< T >();
00155         move_previous(this->end(), list.begin(), list.end());
00156         list.reset();
00157     }
00158
00159     List< T >& operator=(std::initializer_list< T > list) {
00160         assign(list);
00161         return *this;
00162     }
00163
00164     List< T >& operator=(const List< T >& list) {
00165         if (this == &list) return *this;
00166         this->assign(list.begin(), list.end());
00167         return *this;
00168     }
00169
00170     List< T >& operator=(List< T >&& list) {
00171         if (this == &list) return *this;
00172         this->clear();
00173         this->move_previous(this->end(), list.begin(), list.end());
00174         list.reset();
00175         return *this;
00176     }
00177
00178     protected:
00179         iterator begin() { return mBegin; }
00180         const_iterator begin() const { return mBegin; }
00181         iterator end() { return mEnd; }
00182         const_iterator end() const { return mEnd; }
00183
00184     public:
00185         [[nodiscard]] bool empty() const { return this->size() == 0; }
00186
00187         [[nodiscard]] std::size_t size() const { return this->mSize; }
00188
00189     protected:
00190         T& front() { return *this->begin(); }
00191
00192         const T& front() const { return *this->begin(); }
00193
00194         T& back() {
00195             auto it = this->end();
00196             --it;
00197             return *it;
00198         }
00199
00200         const T& back() const {
00201             auto it = this->end();
00202             --it;
00203             return *it;
00204         }
00205
00206         T& operator[](std::size_t index) { return *this->get_iterator(index); }
00207
00208         const T& operator[](std::size_t index) const {
00209             return *this->get_iterator(index);
00210         }
00211
00212         T& at(std::size_t index) { return (*this)[index]; }

```

```

00296
00297     const T& at(std::size_t index) const { return (*this)[index]; }
00298
00299
00300     void push_front(const T& value) {
00301         Node< T >* node = new Node< T >(value);
00302         this->insert_previous(this->begin(), iterator(node));
00303     }
00304
00305
00306     void push_back(const T& value) {
00307         Node< T >* node = new Node< T >(value);
00308         this->insert_previous(this->end(), iterator(node));
00309     }
00310
00311
00312     void pop_front() {
00313         auto it = this->begin();
00314         this->remove(it);
00315     }
00316
00317
00318     void pop_back() {
00319         auto it = --this->end();
00320         this->remove(it);
00321     }
00322
00323
00324     void protected:
00325         iterator remove(const iterator& it) {
00326             if (it == this->end()) {
00327                 throw std::out_of_range("Iterator out of range");
00328             }
00329
00330             Node< T >* node = it.ptr;
00331             Node< T >* node_prev = node->mPrev;
00332             Node< T >* node_next = node->mNext;
00333
00334             if (node_prev != nullptr) node_prev->mNext = node_next;
00335
00336             if (node_next != nullptr) node_next->mPrev = node_prev;
00337
00338             if (it == this->begin()) {
00339                 this->mBegin = node_next;
00340             }
00341
00342             delete node;
00343             --this->mSize;
00344             return node_next;
00345         }
00346
00347
00348         int remove(const T& value, const iterator& begin, const iterator& end) {
00349             return this->remove_if(
00350                 [&value](const T& element) { return element == value; },
00351                 begin,
00352                 end);
00353         }
00354
00355
00356         int remove(const T& value) {
00357             return this->remove_if(
00358                 [&value](const T& element) { return element == value; },
00359                 this->begin(),
00360                 this->end());
00361         }
00362
00363
00364         /*
00365             @brief Removes all elements satisfying specific criteria from the
00366             range [first, last) and returns a past-the-end iterator for the new
00367             end of the range.
00368             @return the resulting size
00369         */
00370         int remove_if(std::function< bool(const T&) > predicate,
00371                     const iterator& begin, const iterator& end) {
00372             for (auto it = begin; it != end; ) {
00373                 if (predicate(*it)) {
00374                     it = this->remove(it);
00375                 } else {
00376                     ++it;
00377                 }
00378             }
00379             return this->size();
00380         }
00381
00382
00383         /*
00384             @brief Removes all elements satisfying specific criteria from the
00385             range [first, last) and returns a past-the-end iterator for the new
00386             end of the range.
00387             @return the resulting size
00388         */
00389         int remove_if(std::function< bool(const T&) > predicate) {
00390             return this->remove_if(predicate, this->begin(), this->end());
00391         }
00392
00393         void resize(std::size_t count) {

```

```
00425     while (this->size() < count) this->push_back(T());
00426
00427     while (this->size() > count) this->pop_back();
00428 }
00429
00430 void resize(std::size_t count, const T& value) {
00431     while (this->size() < count) this->push_back(value);
00432
00433     while (this->size() > count) this->pop_back();
00434 }
00435
00436 void assign(std::size_t count, const T& value) {
00437     this->clear();
00438     this->resize(count, value);
00439 }
00440
00441 void assign(const const_iterator& begin, const const_iterator& end) {
00442     this->resize(std::distance(begin, end));
00443
00444     auto itOther = begin;
00445     for (auto it = this->begin(); it != this->end(); ++it, ++itOther) {
00446         *it = *itOther;
00447     }
00448 }
00449
00450 void assign(std::initializer_list< T > list) {
00451     this->resize(list.size());
00452
00453     auto itOther = list.begin();
00454     for (auto it = this->begin(); it != this->end(); ++it, ++itOther) {
00455         (*it) = (*itOther);
00456     }
00457
00458 std::size_t unique() {
00459     iterator last = std::unique(this->begin(), this->end());
00460     this->resize(std::distance(this->begin(), last));
00461
00462     return this->size();
00463 }
00464
00465 std::size_t unique(
00466     std::function< bool(const T&, const T&) > predicate) {
00467     iterator last = std::unique(this->begin(), this->end(), predicate);
00468     this->resize(std::distance(this->begin(), last));
00469
00470     return this->size();
00471 }
00472
00473 void reverse() {
00474     if (!this->empty()) {
00475         for (auto it = this->begin(), itNext = this->begin();
00476              it != this->endend();
00489     --new_head;
00490     iterator new_tail = this->begin();
00491
00492     new_head.ptr->mPrev = nullptr;
00493     new_tail.ptr->mNext = this->end().ptr;
00494     this->mBegin = new_head;
00495     this->mEnd.ptr->mPrev = new_tail.ptr;
00496 }
00497
00498 protected:
00499     void swap(List< T >& other) {
00500         std::swap(this->mBegin, other.mBegin);
00501         std::swap(this->mEnd, other.mEnd);
00502         std::swap(this->mSize, other.mSize);
00503     }
00504
00505 public:
00506     void clear() {
00507         if (!this->empty()) {
00508             for (auto it = this->begin(); it != this->end();) {
00509                 Node< T *>* node = it.ptr;
00510                 ++it;
00511             }
00512         }
00513     }
00514 }
```

```
00563             delete node;
00564         }
00565     }
00566     this->reset();
00567 }
00568 };
00569 // namespace Core
00570
00571 namespace Core {
00572     template< typename T >
00573     class List< T >::iterator
00574     // : public std::iterator< std::bidirectional_iterator_tag, T,
00575     //   std::ptrdiff_t, T*, T& >
00576     // deprecated since C++17
00577     {
00578         friend class List;
00579
00580     private:
00581         Node< T >* ptr;
00582
00583     public:
00584         using iterator_category = std::bidirectional_iterator_tag;
00585         using value_type = T;
00586         using difference_type = std::ptrdiff_t;
00587         using pointer = T*;
00588         using reference = T&;
00589
00590         iterator() : ptr{nullptr} {}
00591
00592         iterator(Node< value_type >* const p) : ptr(p) {}
00593         reference operator*() const {
00594             reference value = ptr->mValue;
00595             return value;
00596         }
00597
00598         pointer operator->() const { return std::addressof(operator*()); }
00599
00600         iterator& operator++() {
00601             ptr = ptr->mNext;
00602             return *this;
00603         }
00604
00605         iterator operator++(int) {
00606             iterator tmp = *this;
00607             ++(*this);
00608             return tmp;
00609         }
00610
00611         iterator& operator--() {
00612             ptr = ptr->mPrev;
00613             return *this;
00614         }
00615
00616         iterator operator--(int) {
00617             iterator tmp = *this;
00618             --(*this);
00619             return tmp;
00620         }
00621
00622         iterator& operator=(Node< value_type >* const p) {
00623             ptr = p;
00624             return *this;
00625         }
00626
00627         iterator operator+(const int& step) {
00628             iterator it = *this;
00629             if (step > 0) {
00630                 for (int i = 0; i < step; ++i) ++it;
00631             }
00632             if (step < 0) {
00633                 for (int i = 0; i < -step; ++i) --it;
00634             }
00635             return it;
00636         }
00637
00638         iterator operator-(const int& step) {
00639             iterator it = *this;
00640             if (step > 0) {
00641                 for (int i = 0; i < step; ++i) --it;
00642             }
00643             if (step < 0) {
00644                 for (int i = 0; i < -step; ++i) ++it;
00645             }
00646             return it;
00647         }
00648
00649         bool operator==(const iterator& it) const { return it.ptr == ptr; }
```

```
00704     bool operator!=(const iterator& it) const { return it.ptr != ptr; }
00710
00711     void swap(iterator& other) { std::swap(this->ptr, other.ptr); }
00717 }
00718
00723     template< class T >
00724     class List< T >::const_iterator
00725     // : public std::iterator< std::bidirectional_iterator_tag, T,
00726     //           std::ptrdiff_t, T*, T& >
00727     // deprecated since C++17
00728     {
00729         friend class List;
00730
00731     private:
00732         const Node< T >* ptr;
00733
00734     public:
00735         using iterator_category = std::bidirectional_iterator_tag;
00736         using value_type = T;
00737         using difference_type = std::ptrdiff_t;
00738         using pointer = const T*;
00739         using reference = const T&;
00740
00741         const_iterator() : ptr(nullptr) {}
00742
00743         const_iterator(Node< value_type >* const p) : ptr(p) {}
00744
00745         const_iterator(const iterator& other) : ptr(other.ptr) {}
00746
00747         const_iterator(const const_iterator& other) : ptr(other.ptr) {}
00748
00749         reference operator*() const {
00750             reference value = ptr->mValue;
00751             return value;
00752         }
00753
00754         pointer operator->() const { return std::addressof(operator*()); }
00755
00756         const_iterator& operator++() {
00757             ptr = ptr->mNext;
00758             return *this;
00759         }
00760
00761         const_iterator operator++(int) {
00762             const_iterator tmp = *this;
00763             ++(*this);
00764             return tmp;
00765         }
00766
00767         const_iterator& operator--() {
00768             ptr = ptr->mPrev;
00769             return *this;
00770         }
00771
00772         const_iterator operator--(int) {
00773             const_iterator tmp = *this;
00774             --(*this);
00775             return tmp;
00776         }
00777
00778         const_iterator operator+(const int& step) {
00779             const_iterator it = *this;
00780             if (step > 0) {
00781                 for (int i = 0; i < step; ++i) ++it;
00782             }
00783             if (step < 0) {
00784                 for (int i = 0; i < -step; ++i) --it;
00785             }
00786             return it;
00787         }
00788
00789         const_iterator operator-(const int& step) {
00790             const_iterator it = *this;
00791             if (step > 0) {
00792                 for (int i = 0; i < step; ++i) --it;
00793             }
00794             if (step < 0) {
00795                 for (int i = 0; i < -step; ++i) ++it;
00796             }
00797             return it;
00798         }
00799
00800         const_iterator& operator=(Node< value_type >* const p) {
00801             ptr = p;
00802             return *this;
00803         }
```

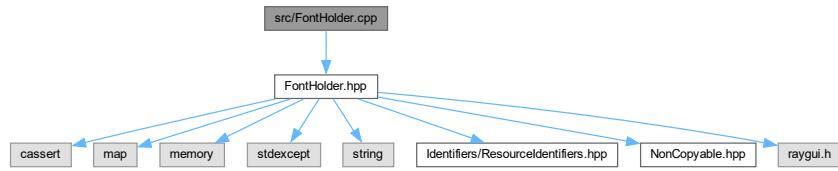
```

00863     const_iterator& operator=(const iterator& other) {
00864         ptr = other.ptr;
00865         return *this;
00866     }
00867
00868     bool operator==(const const_iterator& it) const {
00869         return it.ptr == ptr;
00870     }
00871
00872     bool operator!=(const const_iterator& it) const {
00873         return it.ptr != ptr;
00874     }
00875
00876     void swap(const_iterator& other) { std::swap(this->ptr, other.ptr); }
00877 };
00878 // namespace Core
00879
00880 #endif // CORE_LIST_HPP

```

## 8.100 src/FontHolder.cpp File Reference

#include "FontHolder.hpp"  
Include dependency graph for FontHolder.cpp:



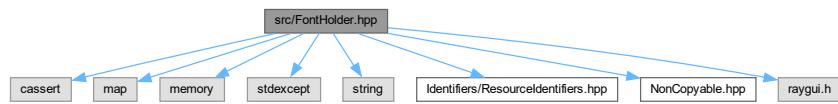
## 8.101 src/FontHolder.hpp File Reference

```

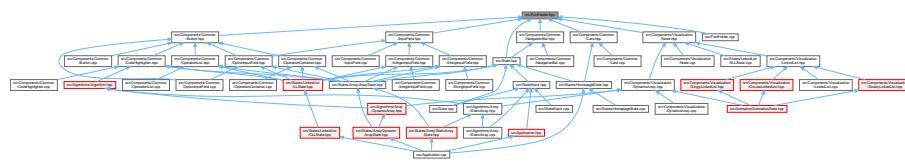
#include <cassert>
#include <map>
#include <memory>
#include <stdexcept>
#include <string>
#include "Identifiers/ResourceIdentifiers.hpp"
#include "NonCopyable.hpp"
#include "raygui.h"

```

Include dependency graph for FontHolder.hpp:



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



## Classes

- class `FontHolder`

*The font holder class that is used to store the fonts.*

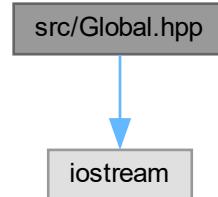
## 8.102 FontHolder.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef FONTHOLDER_HPP
00002 #define FONTHOLDER_HPP
00003
00004 #include <cassert>
00005 #include <map>
00006 #include <memory>
00007 #include <stdexcept>
00008 #include <string>
00009
00010 #include "Identifiers/ResourceIdentifiers.hpp"
00011 #include "NonCopyable.hpp"
00012 #include "raygui.h"
00013
00021 class FontHolder : private NonCopyable< FontHolder > {
00022 public:
00027     FontHolder();
00028
00033     ~FontHolder();
00034
00042     void Load(Fonts::ID id, const std::string& filename);
00043
00051     Font& Get(Fonts::ID id);
00052
00060     const Font& Get(Fonts::ID id) const;
00061
00062 private:
00070     void InsertResource(Fonts::ID id, std::unique_ptr< Font > font);
00071
00072 private:
00079     std::map< Fonts::ID, std::unique_ptr< Font > > mFontMap;
00080 };
00081 #endif // FONTHOLDER_HPP
```

## 8.103 src/Global.hpp File Reference

```
#include <iostream>
Include dependency graph for Global.hpp:
```



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



### Namespaces

- namespace [global](#)

### Variables

- constexpr int [global::SCREEN\\_WIDTH](#) = 1300
- constexpr int [global::SCREEN\\_HEIGHT](#) = 800
- const std::string [global::kTitle](#) = "CS162 - VisuAlgo Clone"
- const std::string [global::favicon](#) = "assets/images/favicon.png"

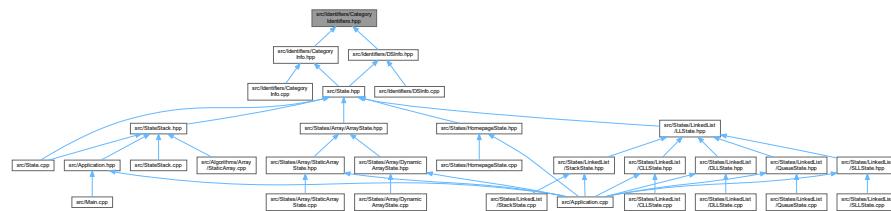
## 8.104 Global.hpp

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include <iostream>
00004
00005 // Global variables
00006 namespace global {
00007     constexpr int SCREEN_WIDTH = 1300;
00008     constexpr int SCREEN_HEIGHT = 800;
00009     // constexpr int kFramesPerSecond = 120;
00010     const std::string kTitle = "CS162 - VisuAlgo Clone";
00011
00012     const std::string favicon = "assets/images/favicon.png";
00013 } // namespace global
```

## 8.105 src/Identifiers/CategoryIdentifiers.hpp File Reference

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



## Namespaces

- namespace Category

## Enumerations

- enum Category::ID { Category::None , Category::Array , Category::LinkedList , Category::Count }

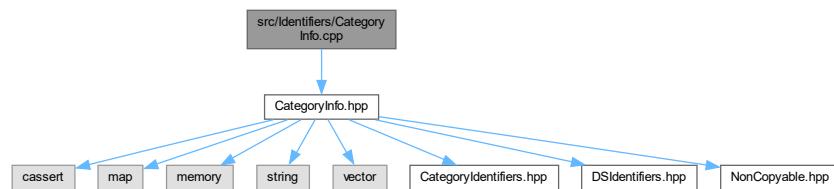
## 8.106 CategoryIdentifiers.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef CATEGORYIDENTIFIERS_HPP
00002 #define CATEGORYIDENTIFIERS_HPP
00003
00004 namespace Category {
00005     enum ID { None, Array, LinkedList, Count };
00006 };
00007
00008 #endif // CATEGORYIDENTIFIERS_HPP
```

## **8.107 src/Identifiers/CategoryInfo.cpp File Reference**

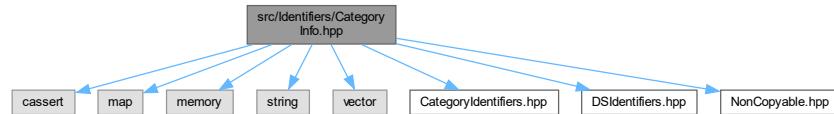
```
#include "CategoryInfo.hpp"  
Include dependency graph for CategoryInfo.cpp:
```



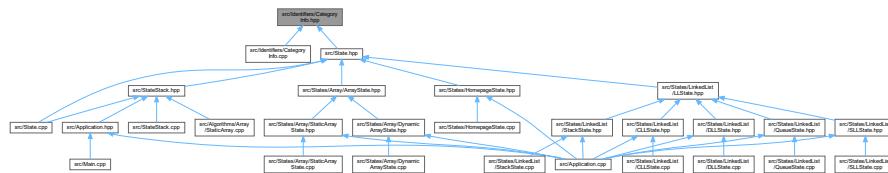
## 8.108 src/Identifiers/CategoryInfo.hpp File Reference

```
#include <cassert>
#include <map>
#include <memory>
#include <string>
#include <vector>
#include "CategoryIdentifiers.hpp"
#include "DSIdentifiers.hpp"
#include "NonCopyable.hpp"
Include dependency graph for CategoryInfo.hpp:
```

Include dependency graph for CategoryInfo.hpp:



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



## Classes

- class [CategoryInfo](#)  
*The category info class that is used to store information about the categories.*
  - struct [CategoryInfo::Info](#)  
*The info struct that is used to store information about the categories.*

## 8.109 CategoryInfo.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef CATEGORYINFO_HPP
00002 #define CATEGORYINFO_HPP
00003
00004 #include <cassert>
00005 #include <map>
00006 #include <memory>
00007 #include <string>
00008 #include <vector>
00009
00010 #include "CategoryIdentifiers.hpp"
00011 #include "DSIdentifiers.hpp"
00012 #include "NonCopyable.hpp"
00013
00025 class CategoryInfo : private NonCopyable< CategoryInfo > {
00026 private:
00034     struct Info {
```

```

00043     Info(Category::ID categoryID, std::vector< DataStructures::ID > mDS,
00044             std::string name);
00045
00050     Category::ID categoryID;
00051
00055     std::string categoryName;
00056
00061     std::vector< DataStructures::ID > mDS;
00062 }
00063
00069     std::map< Category::ID, Info > mFactories;
00070
00071 public:
00080     void Register(Category::ID, Info info);
00081
00092     const Info& Get(Category::ID id) const;
00093 };
00094
00095 #endif // CATEGORYINFO_HPP

```

## 8.110 src/Identifiers/ColorThemelIdentifiers.hpp File Reference

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



### Namespaces

- namespace [ColorTheme](#)

### Enumerations

- enum [ColorTheme::ID](#) {
 [ColorTheme::Background](#) , [ColorTheme::Text](#) , [ColorTheme::Logo1FirstPart](#) , [ColorTheme::Logo1SecondPart](#) ,
 [ColorTheme::Logo2FirstPart](#) , [ColorTheme::Logo2SecondPart](#) , [ColorTheme::NavigationBar\\_SelectedTitle](#) ,
 [ColorTheme::NavigationBar\\_UnselectedTitle](#) ,
 [ColorTheme::NavigationBar\\_Background](#) , [ColorTheme::Footer\\_Background](#) , [ColorTheme::Footer\\_Icon](#) ,
 [ColorTheme::Footer\\_HoveredIcon](#) ,
 [ColorTheme::Button\\_Background](#) , [ColorTheme::Button\\_HoveredBackground](#) , [ColorTheme::Button\\_Text](#) ,
 [ColorTheme::Card\\_Background](#) ,
 [ColorTheme::Card\\_Text](#) , [ColorTheme::ActionList\\_Text](#) , [ColorTheme::ActionList\\_Background](#) , [ColorTheme::ActionList\\_HoverEffect](#) ,
 [ColorTheme::InputField\\_Inactive](#) , [ColorTheme::CodeHighlighter\\_Background](#) , [ColorTheme::CodeHighlighter\\_Text](#) ,
 [ColorTheme::CodeHighlighter\\_HighlightedLineBackground](#) ,
 [ColorTheme::ActionDescription\\_Background](#) , [ColorTheme::ActionDescription\\_Text](#) , [ColorTheme::Visualizer\\_Label](#) ,
 [ColorTheme::Visualizer\\_ErrorText](#) ,
 [ColorTheme::Visualizer\\_ActionText](#) , [ColorTheme::Visualizer\\_Node\\_Default\\_Outline1](#) , [ColorTheme::Visualizer\\_Node\\_Default\\_Outline2](#) ,
 [ColorTheme::Visualizer\\_Node\\_Default\\_Background1](#) ,
 [ColorTheme::Visualizer\\_Node\\_Default\\_Background2](#) , [ColorTheme::Visualizer\\_Node\\_Default\\_Text1](#) ,
 [ColorTheme::Visualizer\\_Node\\_Default\\_Text2](#) , [ColorTheme::Visualizer\\_Node\\_Active\\_Outline1](#) ,
 [ColorTheme::Visualizer\\_Node\\_Active\\_Outline2](#) , [ColorTheme::Visualizer\\_Node\\_Active\\_Background1](#) ,
 [ColorTheme::Visualizer\\_Node\\_Active\\_Background2](#) , [ColorTheme::Visualizer\\_Node\\_Active\\_Text1](#) ,
 [ColorTheme::Visualizer\\_Node\\_Active\\_Text2](#) , [ColorTheme::Visualizer\\_Node\\_ActiveBlue\\_Outline1](#) ,
 [ColorTheme::Visualizer\\_Node\\_ActiveBlue\\_Outline2](#) , [ColorTheme::Visualizer\\_Node\\_ActiveBlue\\_Background1](#)
 }

```
, ColorTheme::Visualizer_Node_ActiveBlue_Background2 , ColorTheme::Visualizer_Node_ActiveBlue_Text1
, ColorTheme::Visualizer_Node_ActiveBlue_Text2 , ColorTheme::Visualizer_Node_ActiveGreen_Outline1 ,
ColorTheme::Visualizer_Node_ActiveGreen_Outline2 , ColorTheme::Visualizer_Node_ActiveGreen_Background1
, ColorTheme::Visualizer_Node_ActiveGreen_Background2 , ColorTheme::Visualizer_Node_ActiveGreen_Text1

,
ColorTheme::Visualizer_Node_ActiveGreen_Text2 , ColorTheme::Visualizer_Node_ActiveRed_Outline1 ,
ColorTheme::Visualizer_Node_ActiveRed_Outline2 , ColorTheme::Visualizer_Node_ActiveRed_Background1

,
ColorTheme::Visualizer_Node_ActiveRed_Background2 , ColorTheme::Visualizer_Node_ActiveRed_Text1 ,
ColorTheme::Visualizer_Node_ActiveRed_Text2 , ColorTheme::Visualizer_Node_Iterated_Outline1 ,
ColorTheme::Visualizer_Node_Iterated_Outline2 , ColorTheme::Visualizer_Node_Iterated_Background1 ,
ColorTheme::Visualizer_Node_Iterated_Background2 , ColorTheme::Visualizer_Node_Iterated_Text1 ,
ColorTheme::Visualizer_Node_Iterated_Text2 , ColorTheme::Visualizer_Arrow_Default , ColorTheme::Visualizer_Arrow_Active
, ColorTheme::Count }
```

## 8.111 ColorThemeldentifiers.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef IDENTIFIERS_COLORIDENTIFIERS_HPP
00002 #define IDENTIFIERS_COLORIDENTIFIERS_HPP
00003
00004 namespace ColorTheme {
00005     enum ID {
00006         /* Background */
00007         Background,
00008         Text,
00009         /* GUI */
00010         /* Logo color */
00011         Logo1FirstPart,
00012         Logo1SecondPart,
00013         Logo2FirstPart,
00014         Logo2SecondPart,
00015
00016         /* Navigation bar */
00017         NavigationBar_SelectedTitle,
00018         NavigationBar_UnselectedTitle,
00019         NavigationBar_Background,
00020         /* Footer */
00021         Footer_Background,
00022         Footer_Icon,
00023         Footer_HoveredIcon,
00024
00025         /* Button */
00026         Button_Background,
00027         Button_HoveredBackground,
00028         Button_Text,
00029
00030         /* Card (title) */
00031         Card_Background,
00032         Card_Text,
00033
00034         /* Action list */
00035         ActionList_Text,
00036         ActionList_Background,
00037         ActionList_HoverBackground,
00038         /* Input */
00039         InputField_Inactive,
00040
00041         /* Code highlighter */
00042         CodeHighlighter_Background,
00043         CodeHighlighter_Text,
00044         CodeHighlighter_HighlightedLineBackground,
00045         ActionDescription_Background,
00046         ActionDescription_Text,
00047
00048         /* Visualizer */
00049         /* Node */
00050         Visualizer_Label,
00051         Visualizer_ErrorText,
00052         Visualizer_ActionText,
00053
00054         /* Default */
00055         Visualizer_Node_Default_Outline1,
```

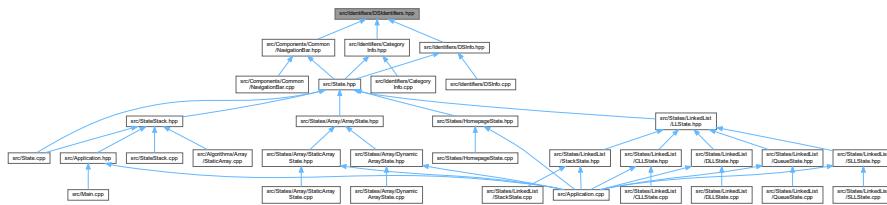
```

00056
00057
00058
00059
00060
00061
00062     /* Active */
00063     Visualizer_Node_Active_Outline1,
00064     Visualizer_Node_Active_Outline2,
00065     Visualizer_Node_Active_Background1,
00066     Visualizer_Node_Active_Background2,
00067     Visualizer_Node_Active_Text1,
00068     Visualizer_Node_Active_Text2,
00069
00070     /* Active Blue */
00071     Visualizer_Node_ActiveBlue_Outline1,
00072     Visualizer_Node_ActiveBlue_Outline2,
00073     Visualizer_Node_ActiveBlue_Background1,
00074     Visualizer_Node_ActiveBlue_Background2,
00075     Visualizer_Node_ActiveBlue_Text1,
00076     Visualizer_Node_ActiveBlue_Text2,
00077
00078     /* Active Green */
00079     Visualizer_Node_ActiveGreen_Outline1,
00080     Visualizer_Node_ActiveGreen_Outline2,
00081     Visualizer_Node_ActiveGreen_Background1,
00082     Visualizer_Node_ActiveGreen_Background2,
00083     Visualizer_Node_ActiveGreen_Text1,
00084     Visualizer_Node_ActiveGreen_Text2,
00085
00086     /* Active Red */
00087     Visualizer_Node_ActiveRed_Outline1,
00088     Visualizer_Node_ActiveRed_Outline2,
00089     Visualizer_Node_ActiveRed_Background1,
00090     Visualizer_Node_ActiveRed_Background2,
00091     Visualizer_Node_ActiveRed_Text1,
00092     Visualizer_Node_ActiveRed_Text2,
00093
00094     /* Iterated */
00095     Visualizer_Node_Iterated_Outline1,
00096     Visualizer_Node_Iterated_Outline2,
00097     Visualizer_Node_Iterated_Background1,
00098     Visualizer_Node_Iterated_Background2,
00099     Visualizer_Node_Iterated_Text1,
00100     Visualizer_Node_Iterated_Text2,
00101
00102     /* Arrow */
00103     Visualizer_Arrow_Default,
00104     Visualizer_Arrow_Active,
00105
00106     Count,
00107 };
00108 };
00109
00110 #endif // IDENTIFIERS_COLORIDENTIFIERS_HPP

```

## 8.112 src/Identifiers/DSIdentifiers.hpp File Reference

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



## Namespaces

- namespace DataStructures

## Enumerations

- enum DataStructures::ID {
 DataStructures::None , DataStructures::StaticArray , DataStructures::DynamicArray , DataStructures::SinglyLinkedList ,
 DataStructures::DoublyLinkedList , DataStructures::CircularLinkedList , DataStructures::Stack , DataStructures::Queue ,
 DataStructures::Count }

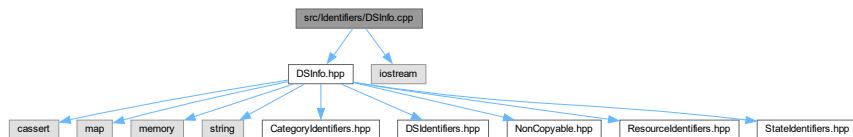
## 8.113 DSIdentifiers.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef DSIDENTIFIERS_HPP
00002 #define DSIDENTIFIERS_HPP
00003
00004 namespace DataStructures {
00005     enum ID {
00006         None,
00007         StaticArray,
00008         DynamicArray,
00009         SinglyLinkedList,
00010         DoublyLinkedList,
00011         CircularLinkedList,
00012         Stack,
00013         Queue,
00014         Count
00015     };
00016 }
00017
00018 #endif // DSIDENTIFIERS_HPP
```

## 8.114 src/Identifiers/DSInfo.cpp File Reference

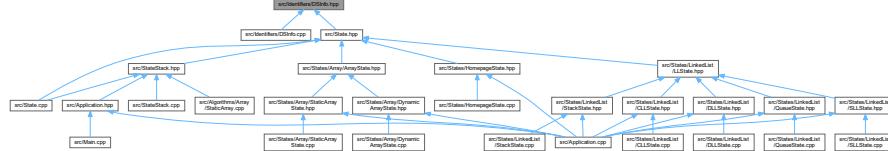
```
#include "DSInfo.hpp"
#include <iostream>
Include dependency graph for DSInfo.cpp:
```



## 8.115 src/Identifiers/DSInfo.hpp File Reference

```
#include <cassert>
#include <map>
#include <memory>
#include <string>
#include "CategoryIdentifiers.hpp"
#include "DSIdentifiers.hpp"
#include "NonCopyable.hpp"
#include "ResourceIdentifiers.hpp"
```

```
#include "StateIdentifiers.hpp"
Include dependency graph for DSInfo.hpp:
```



## Classes

- class [DSInfo](#)  
*The data structure info class that is used to store information about the data structures.*
- struct [DSInfo::Info](#)  
*The info struct that is used to store information about the data structures.*

## 8.116 DSInfo.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef DSINFO_HPP
00002 #define DSINFO_HPP
00003
00004 #include <cassert>
00005 #include <map>
00006 #include <memory>
00007 #include <string>
00008
00009 #include "CategoryIdentifiers.hpp"
00010 #include "DSIdentifiers.hpp"
00011 #include "NonCopyable.hpp"
00012 #include "ResourceIdentifiers.hpp"
00013 #include "StateIdentifiers.hpp"
00014
00029 class DSInfo : private NonCopyable< DSInfo > {
00030     private:
00039         struct Info {
00051             Info(DataStructures::ID ID, States::ID stateID, Category::ID categoryId,
00052                 Textures::ID thumbnail, std::string name, std::string abbr);
00053
00058             DataStructures::ID ID;
00059
00064             States::ID stateID;
00065
00070             Category::ID categoryId;
00071
00076             Textures::ID thumbnail;
00077
00081             std::string name;
00082
00086             std::string abbr;
00087         };
00088
00094     std::map< DataStructures::ID, Info > mFactories;
  
```

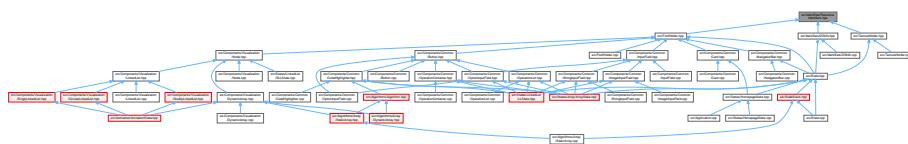
```

00095 public:
00096     void Register(DataStructures::ID, Info info);
00104
00111     Info Get(DataStructures::ID id) const;
00112 };
00113
00114 #endif // DSINFO_HPP

```

## 8.117 src/Identifiers/Resourcelidentifiers.hpp File Reference

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



### Namespaces

- namespace Fonts
- namespace Textures

### Enumerations

- enum Fonts::ID {
 Fonts::Default , Fonts::Default\_Italic , Fonts::Default\_Bold , Fonts::Silkscreen ,
 Fonts::Consolas , Fonts::Courier , Fonts::Courier\_Bold , Fonts::Count }
- enum Textures::ID {
 Textures::Blank , Textures::StaticArray , Textures::DynamicArray , Textures::SinglyLinkedList ,
 Textures::DoublyLinkedList , Textures::CircularLinkedList , Textures::Stack , Textures::Queue ,
 Textures::Favicon , Textures::Count }

## 8.118 Resourcelidentifiers.hpp

[Go to the documentation of this file.](#)

```

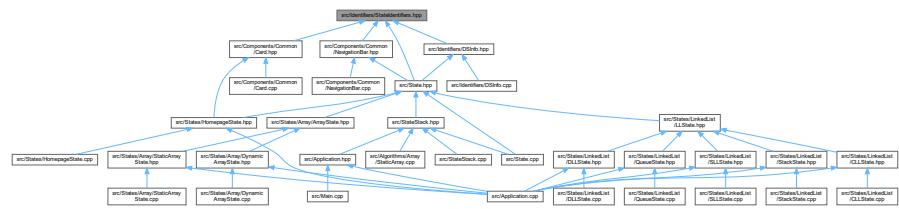
00001 #ifndef RESOURCEIDENTIFIER_HPP
00002 #define RESOURCEIDENTIFIER_HPP
00003
00004 namespace Fonts {
00005     enum ID {
00006         Default,
00007         Default_Italic,
00008         Default_Bold,
00009         Silkscreen,
00010         Consolas,
00011         Courier,
00012         Courier_Bold,
00013         Count
00014     };
00015 };
00016
00017 namespace Textures {
00018     enum ID {
00019         Blank,
00020         StaticArray,
00021         DynamicArray,
00022         SinglyLinkedList,

```

```
00023     DoublyLinkedList,
00024     CircularLinkedList,
00025     Stack,
00026     Queue,
00027     Favicon,
00028     Count
00029 };
00030 };
00031
00032 #endif // RESOURCEIDENTIFIER_HPP
```

## 8.119 src/Identifiers/StatIdentifiers.hpp File Reference

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



# Namespaces

- namespace **States**

## Enumerations

- enum States::ID {  
States::None , States::Homepage , States::StaticArray , States::DynamicArray ,  
States::SinglyLinkedList , States::DoublyLinkedList , States::CircularLinkedList , States::Stack ,  
States::Queue , States::Count }

## 8.120 StatIdentifiers.hpp

[Go to the documentation of this file.](#)

00001 #ifndef STATEIDENTIFIER\_HPP

00002

00003

```
00004 namespace States {
```

```
00005      enum ID {  
00006          None,
```

00006 None,  
00007 Homepage.

00007            homepage,  
00008            StaticArray,

00009 DynamicArray,

00010           SinglyLinkedList,

00011 DoublyLinkedList,  
00012      DoublyLinkedList

00012 CircularLinkedList,  
00013 Stack

00013 Stack,  
00014 Queue

00014 Queue,  
00015 Count

```
00016    };
```

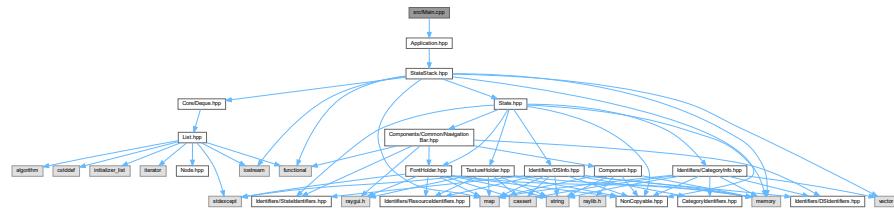
00017 };

00018

```
00019 #endif // STATEIDENTIFIER_HPP
```

## 8.121 src/Main.cpp File Reference

```
#include "Application.hpp"
Include dependency graph for Main.cpp:
```



## Functions

- int [main \(\)](#)

### 8.121.1 Function Documentation

#### 8.121.1.1 main()

```
int main ( )
```

## 8.122 src/NonCopyable.hpp File Reference

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



## Classes

- class [NonCopyable< T >](#)

*The self-explanatory non-copyable class.*

## 8.123 NonCopyable.hpp

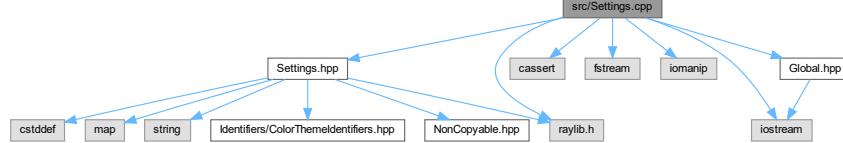
[Go to the documentation of this file.](#)

```
00001 #ifndef NONCOPYABLE_HPP
00002 #define NONCOPYABLE_HPP
00003
00009 template< class T >
00010 class NonCopyable {
00011 public:
00016     NonCopyable(const NonCopyable &) = delete;
00017
00023     T &operator=(const T &) = delete;
00024
00025 protected:
00030     NonCopyable() = default;
00031
00036     ~NonCopyable() = default; // Protected non-virtual destructor
00037 };
00038
00039 #endif // NONCOPYABLE_HPP
```

## 8.124 src/Settings.cpp File Reference

```
#include "Settings.hpp"
#include <cassert>
#include <fstream>
#include <iomanip>
#include <iostream>
#include "Global.hpp"
#include "raylib.h"
```

Include dependency graph for Settings.cpp:



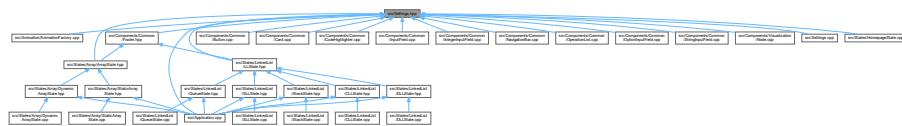
## 8.125 src/Settings.hpp File Reference

```
#include <cstddef>
#include <map>
#include <string>
#include "Identifiers/ColorThemeIdentifiers.hpp"
#include "NonCopyable.hpp"
#include "raylib.h"
```

Include dependency graph for Settings.hpp:



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



## Classes

- class [Settings](#)

*The settings class that is used to store the settings.*

## 8.126 Settings.hpp

[Go to the documentation of this file.](#)

```

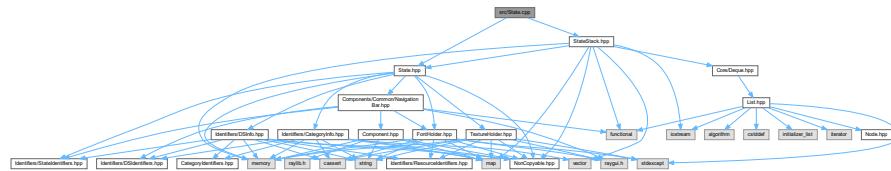
00001 #ifndef SETTINGS_HPP_
00002 #define SETTINGS_HPP_
00003
00004 #include <cstddef>
00005 #include <map>
00006 #include <string>
00007
00008 #include "Identifiers/ColorThemeIdentifiers.hpp"
00009 #include "NonCopyable.hpp"
00010 #include "raylib.h"
0011
0019 class Settings : private NonCopyable< Settings > {
0020 private:
0026     std::size_t mCurrentColorTheme;
0027
0032     std::map< std::size_t, Color > mColors;
0033
0034 private:
0039     Settings() = default;
0040
0041 public:
0048     Settings(Settings&&) = delete;
0049
0057     Settings& operator=(Settings&&) = delete;
0058
0063     ~Settings();
0064
0065 public:
0073     static Settings& getInstance();
0074
0082     Color getColor(std::size_t id) const;
0083
0088     void Load();
0089
0094     void SwitchTheme();
0095
0096 private:
0097     void LoadDefaultColors();
0098     void LoadDarkColors();
0099     // void SaveToFile(const std::string& path);
0100     // void LoadFromFile(const std::string& path);
0101 };
0102
0103 #endif // SETTINGS_HPP_
  
```

## 8.127 src/State.cpp File Reference

```

#include "State.hpp"
#include "StateStack.hpp"
  
```

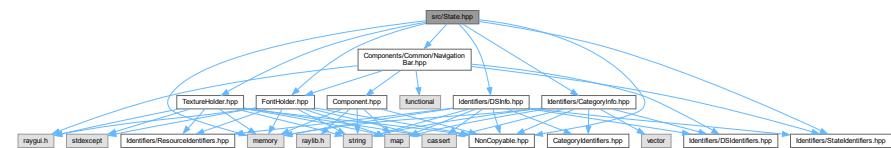
Include dependency graph for State.cpp:



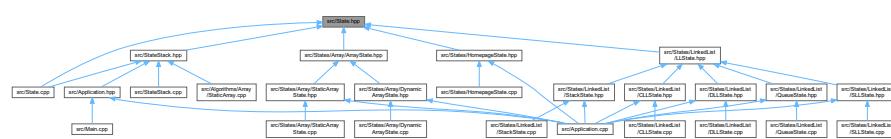
## 8.128 src/State.hpp File Reference

```
#include <memory>
#include "Components/Common/NavigationBar.hpp"
#include "FontHolder.hpp"
#include "Identifiers/StateIdentifiers.hpp"
#include "TextureHolder.hpp"
#include "Identifiers/CategoryInfo.hpp"
#include "Identifiers/DSInfo.hpp"
#include "NonCopyable.hpp"
```

Include dependency graph for State.hpp:



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



## Classes

- class [State::State](#)  
*The base state class that is used to represent a state/scene of the application.*
- struct [State::State::Context](#)  
*The context that is used to share the resources (fonts, textures, ...) between states (scenes).*

## Namespaces

- namespace [State](#)

## 8.129 State.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef STATE_HPP
00002 #define STATE_HPP
00003
00004 #include <memory>
00005
00006 #include "Components/Common/NavigationBar.hpp"
00007 #include "FontHolder.hpp"
00008 #include "Identifiers/StateIdentifiers.hpp"
00009 #include "TextureHolder.hpp"
00010
00011 // Categories and Data Structures identifiers
00012 #include "Identifiers/CategoryInfo.hpp"
00013 #include "Identifiers/DSInfo.hpp"
00014 #include "NonCopyable.hpp"
00015
00016 namespace State {
00017     class StateStack;
00018
00019     class State : private NonCopyable< State > {
00020         public:
00021             typedef std::unique_ptr< State > Ptr;
00022
00023             struct Context {
00024                 Context();
00025                 Context(FontHolder* fonts, TextureHolder* textures,
00026                         CategoryInfo* categories, DSInfo* dsInfo);
00026                 FontHolder* fonts;
00027                 TextureHolder* textures;
00028                 CategoryInfo* categories;
00029                 DSInfo* dsInfo;
00030             };
00031
00032         public:
00033             State(StateStack& stack, Context context);
00034
00035             virtual ~State();
00036
00037             virtual void Draw() = 0;
00038
00039             virtual bool Update(float dt) = 0;
00040
00041         protected:
00042             void RequestStackPush(States::ID stateID);
00043
00044             void RequestStackPop();
00045
00046             void RequestStackClear();
00047
00048             Context GetContext() const;
00049
00050             void InitNavigationBar();
00051
00052         private:
00053             StateStack* mStack;
00054
00055             Context mContext;
00056
00057         protected:
00058             GUIComponent::NavigationBar navigation;
00059         };
00060     }; // namespace State
00061
00062 #endif // STATE_HPP

```

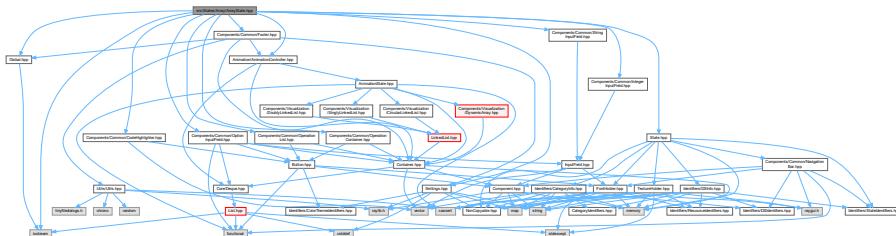
## 8.130 src/States/Array/ArrayType.hpp File Reference

```

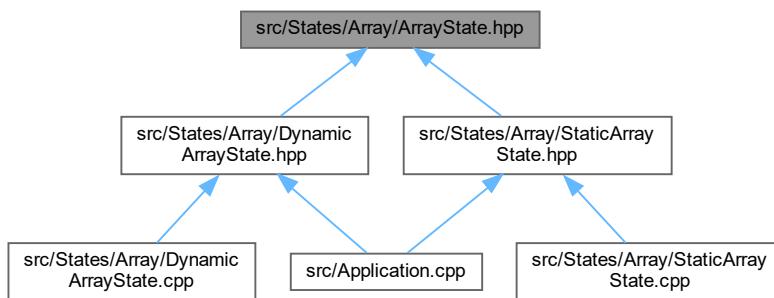
#include "Animation/AnimationController.hpp"
#include "Components/Common/CodeHighlighter.hpp"
#include "Components/Common/Footer.hpp"
#include "Components/Common/OperationContainer.hpp"
#include "Components/Common/OperationList.hpp"
#include "Components/Common/OptionInputField.hpp"
#include "Global.hpp"

```

```
#include "Settings.hpp"
#include "State.hpp"
#include <iostream>
#include "Components/Common/IntegerField.hpp"
#include "Components/Common/StringField.hpp"
Include dependency graph for ArrayState.hpp:
```



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



## Classes

- class `State::ArrayState< T >`  
*The state class that is used as a base array state for all array state/scene of the application.*
- struct `State::ArrayState< T >::IntegerField`

## Namespaces

- namespace `State`

## 8.131 ArrayState.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef STATES_LINKEDLIST_ARRAYSTATE_HPP
00002 #define STATES_LINKEDLIST_ARRAYSTATE_HPP
00003
00004 #include "Animation/AnimationController.hpp"
00005 #include "Components/Common/CodeHighlighter.hpp"
```

```
00006 #include "Components/Common/Footer.hpp"
00007 #include "Components/Common/OperationContainer.hpp"
00008 #include "Components/Common/OperationList.hpp"
00009 #include "Components/Common/OptionInputField.hpp"
00010 #include "Global.hpp"
00011 #include "Settings.hpp"
00012 #include "State.hpp"
00013
00014 namespace State {
00015     template< typename T >
00016     class ArrayState : public State {
00017     public:
00018         struct IntegerInput {
00019             std::string label;
00020             int width;
00021             int minValue;
00022             int maxValue;
00023         };
00024
00025     public:
00026         ArrayState(StateStack& stack, Context context,
00027                     DataStructures::ID activeDS);
00028
00029         ~ArrayState();
00030
00031         virtual void Draw();
00032
00033         virtual bool Update(float dt);
00034
00035         virtual void SetCurrentAction(std::string action);
00036
00037         virtual void SetCurrentError(std::string error);
00038
00039         virtual void ClearError();
00040
00041         virtual void ClearAction();
00042
00043         virtual void Success();
00044
00045     protected:
00046         virtual void DrawCurrentActionText();
00047         virtual void DrawCurrentErrorText();
00048
00049     protected:
00050         void InitNavigationBar();
00051         Context mContext;
00052
00053     protected:
00054         GUIComponent::CodeHighlighter::Ptr codeHighlighter;
00055         GUIComponent::Footer< T > footer;
00056         std::string mCurrentAction;
00057         std::string mCurrentError;
00058
00059     protected:
00060         typename T::Ptr animController;
00061
00062     protected:
00063         GUIComponent::OperationList operationList;
00064
00065         virtual void AddOperations(); // DO NOT OVERRIDE THIS FUNCTION
00066
00067         virtual void AddInitializeOperation();
00068
00069         virtual void AddInsertOperation();
00070
00071         virtual void AddDeleteOperation();
00072
00073         virtual void AddUpdateOperation();
00074
00075         virtual void AddSearchOperation();
00076
00077         virtual void AddAccessOperation();
00078
00079     protected:
00080         virtual void AddNoFieldOperationOption(
00081             GUIComponent::OperationContainer::Ptr container, std::string title,
00082             std::function< void() > action);
00083         virtual void AddIntegerFieldOption(
00084             GUIComponent::OperationContainer::Ptr container, std::string title,
00085             Core::Deque< IntegerInput > fields,
00086             std::function< void(std::map< std::string, std::string >) > action);
00087         virtual void AddStringFieldOption(
00088             GUIComponent::OperationContainer::Ptr container, std::string title,
00089             std::string label,
00090             std::function< void(std::map< std::string, std::string >) > action);
00091
00092     private:
```

```

00155     DataStructures::ID activeDS;
00156 };
00157 }; // namespace State
00158
00159 #include <iostream>
00160
00161 #include "Components/Common/IntegerField.hpp"
00162 #include "Components/Common/StringField.hpp"
00163 #include "Global.hpp"
00164
00165 template< typename T >
00166 State::ArrayState< T >::ArrayState(StateStack& stack, Context context,
00167                                     DataStructures::ID activeDS)
00168     : State(stack, context), activeDS(activeDS),
00169     codeHighlighter(new GUIComponent::CodeHighlighter(context.fonts)),
00170     footer(GUIComponent::Footer< T >()), animController(new T()) {
00171     InitNavigationBar();
00172     operationList = GUIComponent::OperationList(context.fonts);
00173     codeHighlighter->SetPosition(global::SCREEN_WIDTH - 40,
00174                                   global::SCREEN_HEIGHT - 334);
00175     codeHighlighter->InitButtons();
00176
00177     footer.SetPosition(0, global::SCREEN_HEIGHT - 40);
00178 }
00179
00180 template< typename T >
00181 State::ArrayState< T >::~ArrayState() {}
00182
00183 template< typename T >
00184 inline void State::ArrayState< T >::Draw() {
00185     const Color sidebarColor =
00186         Settings::getInstance().getColor(ColorTheme::NavigationBar_Background);
00187     DrawRectangle(0, 0, 40, global::SCREEN_HEIGHT, sidebarColor);
00188
00189     DrawRectangle(global::SCREEN_WIDTH - 40, 0, 40, global::SCREEN_HEIGHT,
00190                  sidebarColor);
00191
00192     operationList.Draw();
00193     navigation.Draw();
00194
00195     animController->GetAnimation().Draw();
00196     codeHighlighter->Draw();
00197     footer.Draw(animController.get());
00198     DrawCurrentActionText();
00199     DrawCurrentErrorText();
00200 }
00201
00202 template< typename T >
00203 bool State::ArrayState< T >::Update(float dt) {
00204     animController->Update(dt);
00205     codeHighlighter->Highlight(
00206         animController->GetAnimation().GetHighlightedLine());
00207     codeHighlighter->AddActionDescription(
00208         animController->GetAnimation().GetActionDescription());
00209     return true;
00210 }
00211
00212 template< typename T >
00213 inline void State::ArrayState< T >::SetCurrentAction(std::string action) {
00214     mCurrentAction = action;
00215 }
00216
00217 template< typename T >
00218 inline void State::ArrayState< T >::SetCurrentError(std::string error) {
00219     mCurrentError = error;
00220 }
00221
00222 template< typename T >
00223 inline void State::ArrayState< T >::ClearError() {
00224     mCurrentError.clear();
00225 }
00226
00227 template< typename T >
00228 inline void State::ArrayState< T >::ClearAction() {
00229     mCurrentAction.clear();
00230 }
00231
00232 template< typename T >
00233 inline void State::ArrayState< T >::Success() {
00234     operationList.ToggleOperations();
00235     SetMouseCursor(MOUSE_CURSOR_DEFAULT);
00236     ClearError();
00237 }
00238
00239 template< typename T >
00240 inline void State::ArrayState< T >::DrawCurrentActionText() {
00241     const Color textColor = Settings::getInstance().getColor(ColorTheme::Text);

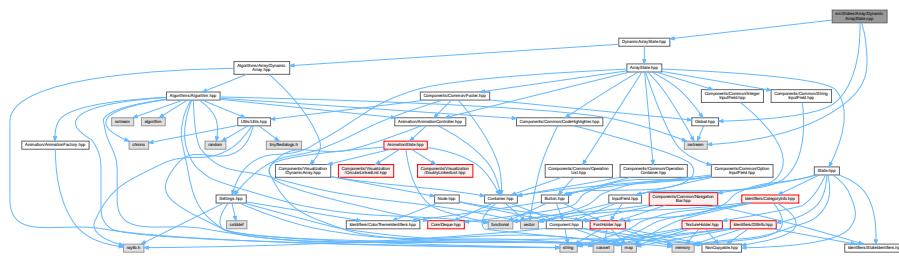
```

```
00242
00243     float rightAlignment = global::SCREEN_WIDTH - 50;
00244     Font font = GetContext().fonts->Get(FONT::Default_Bold);
00245     float width = MeasureTextEx(font, mCurrentAction.c_str(), 32, 0).x;
00246     float x = rightAlignment - width;
00247
00248     DrawTextEx(font, mCurrentAction.c_str(),
00249                 (Vector2){x, global::SCREEN_HEIGHT - 376}, 32, 0, textColor);
00250
00251 // codeHighlighter->SetPosition(, global::SCREEN_HEIGHT - 334);
00252 }
00253
00254 template< typename T >
00255 inline void State::ArrayState< T >::DrawCurrentErrorText() {
00256     const Color errorTextColor =
00257         Settings::GetInstance().getColor(ColorTheme::Visualizer_ErrorText);
00258
00259     Font font = GetContext().fonts->Get(FONT::Default_Bold);
00260     float width = MeasureTextEx(font, mCurrentError.c_str(), 24, 0).x;
00261     float x = 50;
00262
00263     DrawTextEx(
00264         font, mCurrentError.c_str(),
00265         (Vector2){x, global::SCREEN_HEIGHT - operationList.GetSize().y - 90},
00266         24, 0, errorTextColor);
00267 }
00268
00269 template< typename T >
00270 void State::ArrayState< T >::InitNavigationBar() {
00271     navigation.SetVisibleTitle(true);
00272     auto info = GetContext().categories->Get(CATEGORY::Array);
00273     navigation.SetCategory(info.categoryName);
00274     navigation.SetActiveTitle(activeDS);
00275     for (auto dsID : info.mDS) {
00276         auto dsInfo = GetContext().dsInfo->Get(dsID);
00277         navigation.InsertTitle(dsID, dsInfo.stateID, dsInfo.abbr, dsInfo.name);
00278     };
00279 }
00280
00281 template< typename T >
00282 void State::ArrayState< T >::AddOperations() {
00283     AddInitializeOperation();
00284     AddInsertOperation();
00285     AddDeleteOperation();
00286     AddAccessOperation();
00287     AddUpdateOperation();
00288     AddSearchOperation();
00289
00290     operationList.setPosition(
00291         0, global::SCREEN_HEIGHT - 60 - operationList.GetSize().y);
00292     operationList.InitActionBar();
00293 }
00294
00295 template< typename T >
00296 void State::ArrayState< T >::AddInitializeOperation() {}
00297
00298 template< typename T >
00299 void State::ArrayState< T >::AddInsertOperation() {}
00300
00301 template< typename T >
00302 void State::ArrayState< T >::AddDeleteOperation() {}
00303
00304 template< typename T >
00305 void State::ArrayState< T >::AddUpdateOperation() {}
00306
00307 template< typename T >
00308 void State::ArrayState< T >::AddSearchOperation() {}
00309
00310 template< typename T >
00311 inline void State::ArrayState< T >::AddAccessOperation() {}
00312
00313 template< typename T >
00314 void State::ArrayState< T >::AddNoFieldOperationOption(
00315     GUIComponent::OperationContainer::Ptr container, std::string title,
00316     std::function< void() > action) {
00317     GUIComponent::OptionInputField::Ptr button(
00318         new GUIComponent::OptionInputField(GetContext().fonts));
00319
00320     button.get()->SetNoFieldOption(title, action);
00321
00322     container.get()->pack(button);
00323 }
00324
00325 template< typename T >
00326 void State::ArrayState< T >::AddIntFieldOperationOption(
00327     GUIComponent::OperationContainer::Ptr container, std::string title,
00328     Core::Deque< IntegerInput > fields,
```

```
00329     std::function< void(std::map< std::string, std::string >) > action) {
00330     GUIComponent::OptionInputField::Ptr button(
00331         new GUIComponent::OptionInputField(getContext().fonts));
00332     Core::Deque< GUIComponent::InputField::Ptr > intFields;
00333     for (auto field : fields) {
00334         GUIComponent::IntegerInputField::Ptr intField(
00335             new GUIComponent::IntegerInputField(getContext().fonts));
00336         intField.get()->SetLabel(field.label);
00337         intField.get()->SetInputFieldSize((Vector2){(float)field.width, 30});
00338         intField.get()->SetConstraint(field.minValue, field.maxValue);
00339         intField.get()->Randomize();
00340         intFields.push_back(intField);
00341     }
00342     button.get()->SetOption(title, intFields, action);
00343     container.get()->pack(button);
00344 }
00345
00346 }
00347
00348 template< typename T >
00349 void State::ArrayState< T >::AddStringFieldOption(
00350     GUIComponent::OperationContainer::Ptr container, std::string title,
00351     std::string label,
00352     std::function< void(std::map< std::string, std::string >) > action) {
00353     GUIComponent::OptionInputField::Ptr button(
00354         new GUIComponent::OptionInputField(getContext().fonts));
00355     GUIComponent::StringInputField::Ptr strField(
00356         new GUIComponent::StringInputField(getContext().fonts));
00357     strField.get()->SetLabel(label);
00358     strField.get()->SetInputFieldSize((Vector2){100, 30});
00359     button.get()->SetOption(title, {strField}, action);
00360     container.get()->pack(button);
00361 }
00362
00363 #endif // STATES_LINKEDLIST_ARRAYSTATE_HPP
```

## 8.132 src/States/Array/DynamicArrayState.cpp File Reference

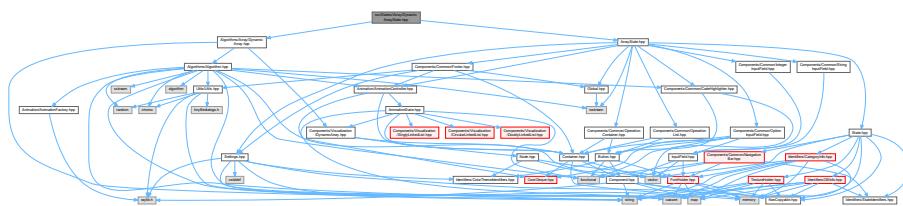
```
#include "DynamicArrayState.hpp"
#include <iostream>
#include "Global.hpp"
Include dependency graph for DynamicArrayState.cpp:
```



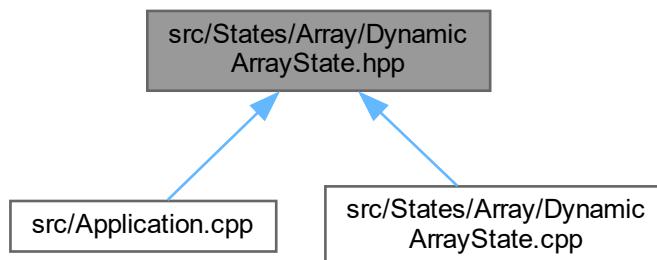
## 8.133 src/States/Array/DynamicArrayState.hpp File Reference

```
#include "Algorithms/Array/DynamicArray.hpp"
#include "ArrayState.hpp"
```

Include dependency graph for DynamicArrayState.hpp:



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



## Classes

- class [State::DynamicArrayState](#)

*The state class that is used to represent the dynamic array state/scene of the application.*

## Namespaces

- namespace [State](#)

## 8.134 DynamicArrayState.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef STATES_ARRAY_DYNAMICARRAYSTATE_HPP
00002 #define STATES_ARRAY_DYNAMICARRAYSTATE_HPP
00003
00004 #include "Algorithms/Array/DynamicArray.hpp"
00005 #include "ArrayState.hpp"
00006
00007 namespace State {
00012     class DynamicArrayState : public ArrayState< DArrayAnimationController > {
00013     private:
00017         Algorithm::DynamicArray mDynamicArray;
00018
00019     private:
00023         void AddInsertOperation();
00024
00028         void AddInitializeOperation();

```

```
00029
00033     void AddUpdateOperation();
00034
00038     void AddDeleteOperation();
00039
00043     void AddSearchOperation();
00044
00048     void AddAccessOperation();
00049
00050 public:
00051     DynamicArrayState(StateStack& stack, Context context);
00052
00053     ~DynamicArrayState();
00054 };
00055 // namespace State
00056
00066 #endif // STATES_ARRAY_DYNAMICARRAYSTATE_HPP
```

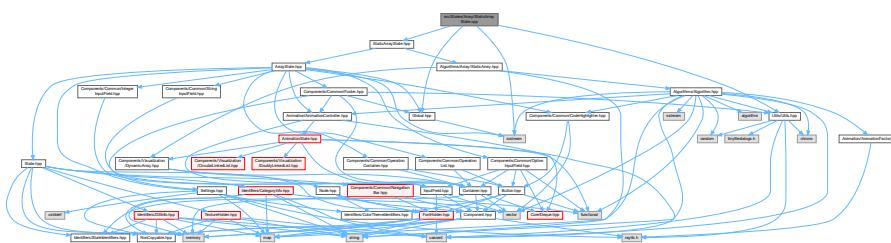
## 8.135 src/States/Array/StaticArrayState.cpp File Reference

```
#include "StaticArrayState.hpp"
#include <iostream>
```

```
#include "Global.hpp"
```

```
#include "Utils/Utils.hpp"
```

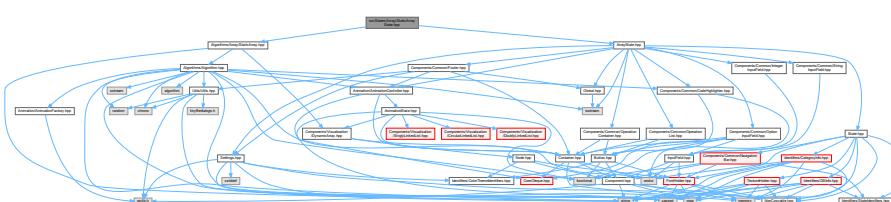
Include dependency graph for StaticArrayState.cpp:



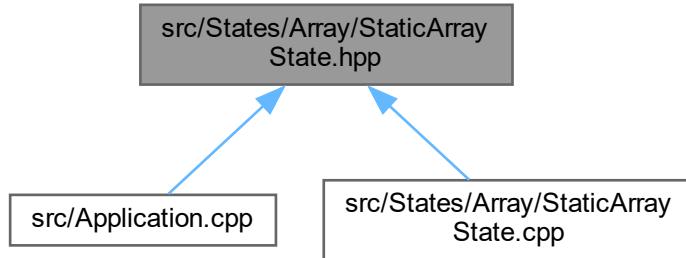
## 8.136 src/States/Array/StaticArrayState.hpp File Reference

```
#include "Algorithms/Array/StaticArray.hpp"
#include "ArrayState.hpp"
```

Include dependency graph for StaticArrayState.hpp:



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



## Classes

- class [State::StaticArrayState](#)

*The state class that is used to represent the static array state/scene of the application.*

## Namespaces

- namespace [State](#)

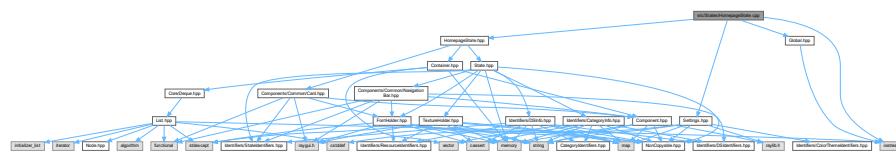
## 8.137 StaticArrayState.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef STATES_ARRAY_STATICARRAYSTATE_HPP
00002 #define STATES_ARRAY_STATICARRAYSTATE_HPP
00003
00004 #include "Algorithms/Array/StaticArray.hpp"
00005 #include "ArrayState.hpp"
00006
00007 namespace State {
00016     class StaticArrayState : public ArrayState< DArrayAnimationController > {
00017     private:
00021         Algorithm::StaticArray mStaticArray;
00022
00023     private:
00027         void AddInsertOperation();
00028
00032         void AddInitializeOperation();
00033
00037         void AddUpdateOperation();
00038
00042         void AddDeleteOperation();
00043
00047         void AddSearchOperation();
00048
00052         void AddAccessOperation();
00053
00054     public:
00061         StaticArrayState(StateStack& stack, Context context);
00062
00066         ~StaticArrayState();
00067     };
00068 }; // namespace State
00069
00070 #endif // STATES_ARRAY_STATICARRAYSTATE_HPP
```

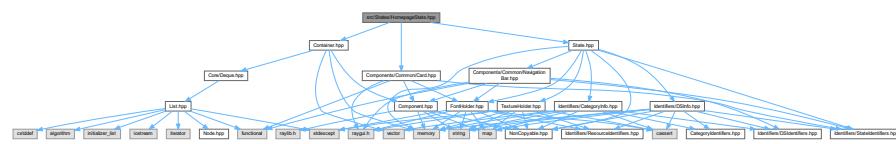
## 8.138 src/States/HomepageState.cpp File Reference

```
#include "HomepageState.hpp"
#include <iostream>
#include "Global.hpp"
#include "Settings.hpp"
Include dependency graph for HomepageState.cpp:
```

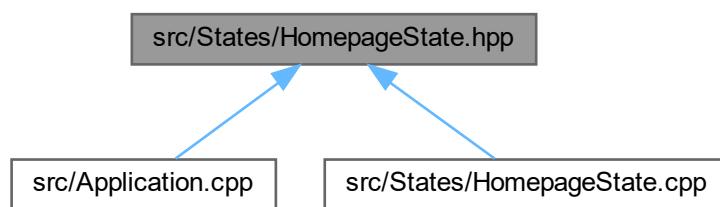


## 8.139 src/States/HomepageState.hpp File Reference

```
#include "Container.hpp"
#include "Components/Common/Card.hpp"
#include "State.hpp"
Include dependency graph for HomepageState.hpp:
```



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



## Classes

- class [State::HomepageState](#)

*The homepage screen.*

## Namespaces

- namespace **State**

## 8.140 HomepageState.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef STATES_HOMEPAGESTATE_HPP
00002 #define STATES_HOMEPAGESTATE_HPP
00003
00004 #include "Container.hpp"
00005 // #include "../Core/Operations/Create/Create.hpp"
00006 #include "Components/Common/Card.hpp"
00007 #include "State.hpp"
00008
00009 namespace State {
00010     class HomepageState : public State {
00011     private:
00012     public:
00013         HomepageState(StateStack& stack, Context context);
00014
00015         ~HomepageState();
00016
00017         void Draw();
00018
00019         bool Update(float dt);
00020
00021     private:
00022         void DrawIntroduction();
00023         void InitCards();
00024         void CreateCard(States::ID stateID, std::string title,
00025                         Textures::ID textureID, int x, int y);
00026
00027     private:
00028         GUI::Container mCards;
00029         bool hasInitializeCard;
00030     };
00031 }; // namespace State
00032
00033 #endif // STATES_HOMEPAGESTATE_HPP

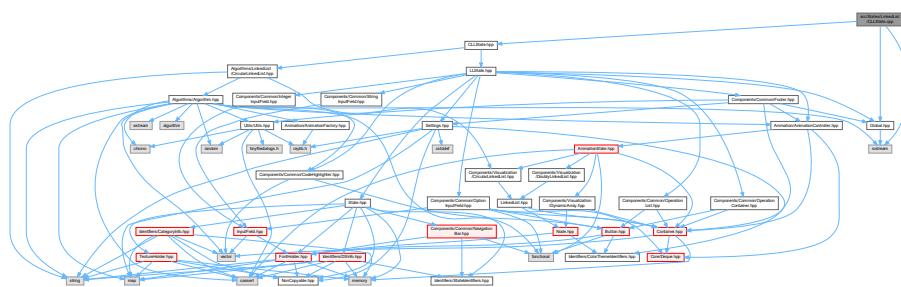
```

## 8.141 src/States/LinkedList/CLLState.cpp File Reference

```

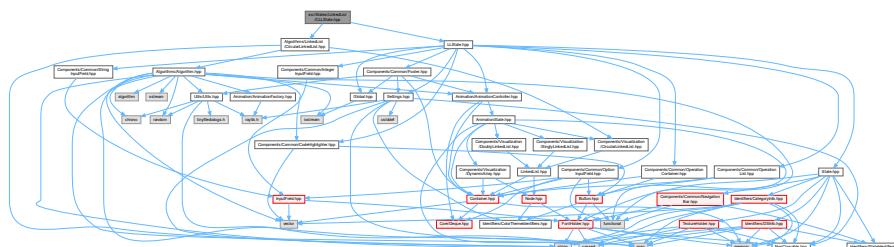
#include "CLLState.hpp"
#include <iostream>
#include "Global.hpp"
Include dependency graph for CLLState.cpp:

```

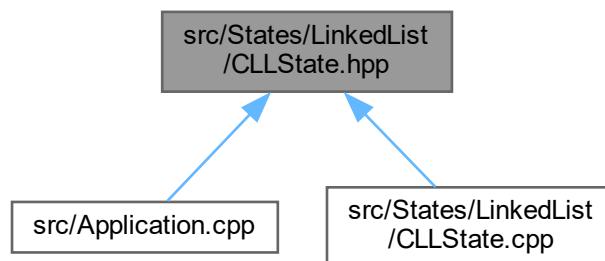


## 8.142 src/States/LinkedList/CLLState.hpp File Reference

```
#include "Algorithms/LinkedList/CircularLinkedList.hpp"
#include "LLState.hpp"
Include dependency graph for CLLState.hpp:
```



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



### Classes

- class [State::CLLState](#)

*The state class that is used to represent the circular linked list state/scene of the application.*

### Namespaces

- namespace [State](#)

## 8.143 CLLState.hpp

[Go to the documentation of this file.](#)

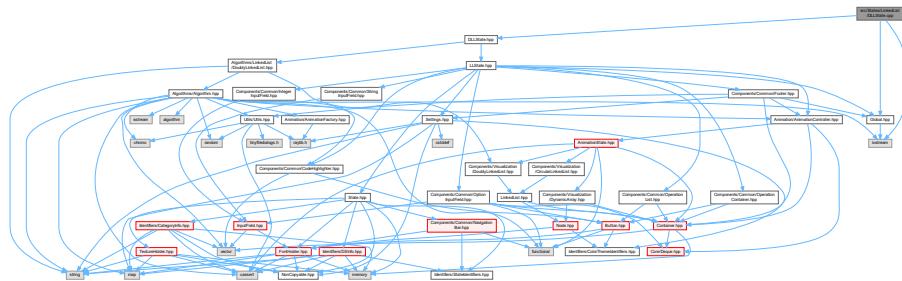
```
00001 #ifndef STATES_LINKEDLIST_CLLSTATE_HPP
00002 #define STATES_LINKEDLIST_CLLSTATE_HPP
00003
00004 #include "Algorithms/LinkedList/CircularLinkedList.hpp"
00005 #include "LLState.hpp"
00006
```

```
00007 namespace State {
00012     class CLLState : public LLState< CLLAnimationController > {
00013     private:
00017         Algorithm::CircularLinkedList CLL;
00018
00019     public:
00023         void AddInsertOperation();
00024
00029         void AddInitializeOperation();
00030
00034         void AddUpdateOperation();
00035
00039         void AddDeleteOperation();
00040
00044         void AddSearchOperation();
00045
00046     public:
00053         CLLState(StateStack& stack, Context context);
00054
00058         ~CLLState();
00059     };
00060 }; // namespace State
00061
00062 #endif // STATES_LINKEDLIST_CLLSTATE_HPP
```

## 8.144 src/States/LinkedList/DLLState.cpp File Reference

```
#include "DLLState.hpp"
#include <iostream>
#include "Global.hpp"
Include dependency graph for DLLS
```

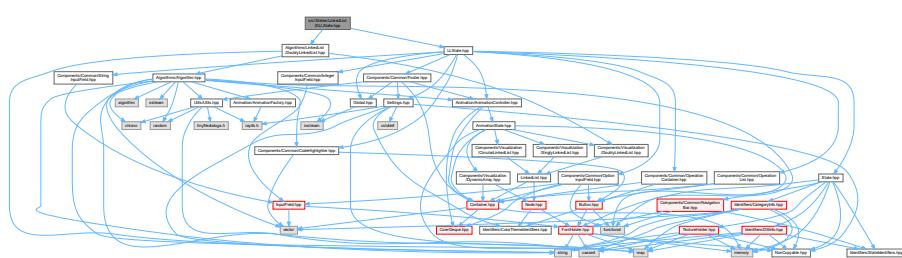
Include dependency graph for DLLState.cpp:



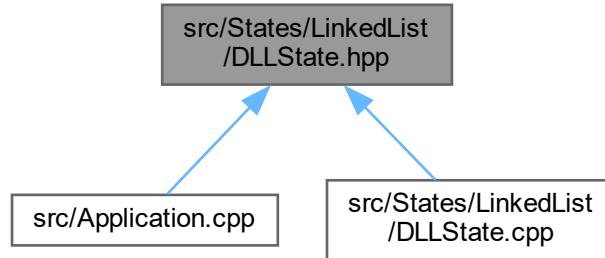
## 8.145 src/States/LinkedList/DLLState.hpp File Reference

```
#include "Algorithms/LinkedList/DoublyLinkedList.hpp"
#include "LLState.hpp"
```

Include dependency graph for DLLState.hpp:



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



## Classes

- class [State::DLLState](#)  
*The state that is used to visualize the doubly linked list.*

## Namespaces

- namespace [State](#)

## 8.146 DLLState.hpp

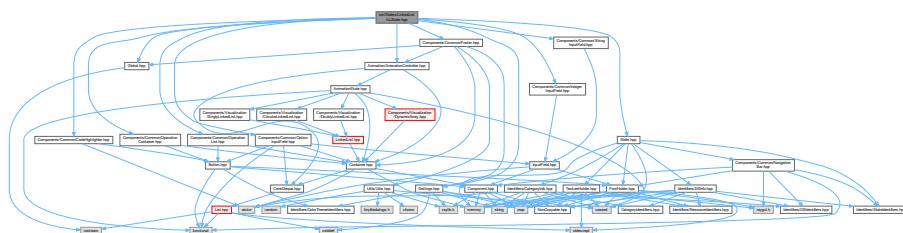
[Go to the documentation of this file.](#)

```

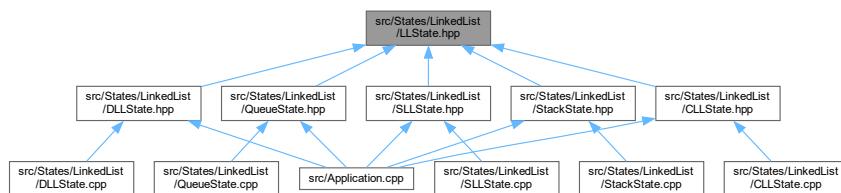
00001 #ifndef STATES_LINKEDLIST_DLLSTATE_HPP
00002 #define STATES_LINKEDLIST_DLLSTATE_HPP
00003
00004 #include "Algorithms/LinkedList/DoublyLinkedList.hpp"
00005 #include "LLState.hpp"
00006
00007 namespace State {
00011     class DLLState : public LLState< DLLAnimationController > {
00012     private:
00016         Algorithm::DoublyLinkedList mDLL;
00017
00018     public:
00022         void AddInsertOperation();
00023
00028         void AddInitializeOperation();
00029
00033         void AddUpdateOperation();
00034
00038         void AddDeleteOperation();
00039
00043         void AddSearchOperation();
00044
00045     public:
00052         DLLState(StateStack& stack, Context context);
00053
00057         ~DLLState();
00058     };
00059 }; // namespace State
00060
00061 #endif // STATES_LINKEDLIST_DLLSTATE_HPP
  
```

## 8.147 src/States/LinkedList/LLState.hpp File Reference

```
#include "Animation/AnimationController.hpp"
#include "Components/Common/CodeHighlighter.hpp"
#include "Components/Common/Footer.hpp"
#include "Components/Common/OperationContainer.hpp"
#include "Components/Common/OperationList.hpp"
#include "Components/Common/OptionInputField.hpp"
#include "Global.hpp"
#include "Settings.hpp"
#include "State.hpp"
#include "Components/Common/IntegerInputField.hpp"
#include "Components/Common/StringInputField.hpp"
Include dependency graph for LLState.hpp:
```



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



## Classes

- class `State::LLState< T >`  
*The state class that is used as a base linked list state/scene of the application.*
- struct `State::LLState< T >::IntegerInput`

## Namespaces

- namespace `State`

## 8.148 LLState.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef STATES_LINKEDLIST_LLSTATE_HPP
00002 #define STATES_LINKEDLIST_LLSTATE_HPP
00003
00004 #include "Animation/AnimationController.hpp"
00005 #include "Components/Common/CodeHighlighter.hpp"
00006 #include "Components/Common/Footer.hpp"
00007 #include "Components/Common/OperationContainer.hpp"
00008 #include "Components/Common/OperationList.hpp"
00009 #include "Components/Common/OptionInputField.hpp"
00010 #include "Global.hpp"
00011 #include "Settings.hpp"
00012 #include "State.hpp"
00013
00014 namespace State {
00015     template< typename T >
00016     class LLState : public State {
00017     public:
00018         struct IntegerInput {
00019             std::string label;
00020             int width;
00021             int minValue;
00022             int maxValue;
00023         };
00024
00025     public:
00026         LLState(StateStack& stack, Context context,
00027                 DataStructures::ID activeDS);
00028
00029         ~LLState();
00030
00031         virtual void Draw();
00032
00033         virtual bool Update(float dt);
00034
00035         virtual void SetCurrentAction(std::string action);
00036
00037         virtual void SetCurrentError(std::string error);
00038
00039         virtual void ClearError();
00040
00041         virtual void ClearAction();
00042
00043         virtual void Success();
00044
00045     protected:
00046         virtual void DrawCurrentActionText();
00047         virtual void DrawCurrentErrorText();
00048
00049     protected:
00050         void InitNavigationBar();
00051         Context mContext;
00052
00053     protected:
00054         GUIComponent::CodeHighlighter::Ptr codeHighlighter;
00055         GUIComponent::Footer< T > footer;
00056         std::string mCurrentAction;
00057         std::string mCurrentError;
00058
00059     protected:
00060         typename T::Ptr animController;
00061
00062     protected:
00063         GUIComponent::OperationList operationList;
00064
00065         virtual void AddOperations(); // DO NOT OVERRIDE THIS FUNCTION
00066
00067         virtual void AddInitializeOperation();
00068
00069         virtual void AddInsertOperation();
00070
00071         virtual void AddDeleteOperation();
00072
00073         virtual void AddUpdateOperation();
00074
00075         virtual void AddSearchOperation();
00076
00077     protected:
00078         virtual void AddNoFieldOperationOption(
00079             GUIComponent::OperationContainer::Ptr container, std::string title,
00080             std::function< void() > action);
00081         virtual void AddIntFieldOperationOption(
00082             GUIComponent::OperationContainer::Ptr container, std::string title,
00083             std::function< void(int) > action);
00084
00085     };
00086 }
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142

```

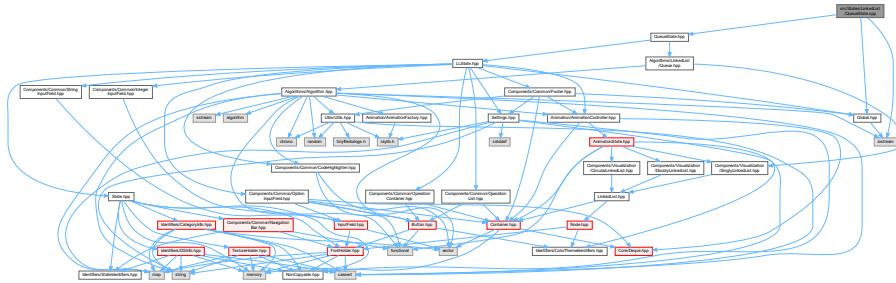
```
00143     Core::Deque< IntegerInput > fields,
00144     std::function< void(std::map< std::string, std::string >) > action);
00145     virtual void AddStringFieldOption(
00146         GUIComponent::OperationContainer::Ptr container, std::string title,
00147         std::string label,
00148         std::function< void(std::map< std::string, std::string >) > action);
00149
00150     private:
00151         DataStructures::ID activeDS;
00152     };
00153 }; // namespace State
00154
00155 #include "Components/Common/IntegerField.hpp"
00156 #include "Components/Common/StringField.hpp"
00157 #include "Global.hpp"
00158
00159 template< typename T >
00160 State::LLState< T >::LLState(StateStack& stack, Context context,
00161                               DataStructures::ID activeDS)
00162 : State(stack, context), activeDS(activeDS),
00163   codeHighlighter(new GUIComponent::CodeHighlighter(context.fonts)),
00164   footer(GUIComponent::Footer< T >()), animController(new T()) {
00165     InitNavigationBar();
00166     operationList = GUIComponent::OperationList(context.fonts);
00167     codeHighlighter->SetPosition(global::SCREEN_WIDTH - 40,
00168                                 global::SCREEN_HEIGHT - 334);
00169     codeHighlighter->InitButtons();
00170
00171     footer.SetPosition(0, global::SCREEN_HEIGHT - 40);
00172 }
00173
00174 template< typename T >
00175 State::LLState< T >::~LLState() {}
00176
00177 template< typename T >
00178 inline void State::LLState< T >::Draw() {
00179     const Color sidebarColor =
00180         Settings::getInstance().getColor(ColorTheme::NavigationBar_Background);
00181     DrawRectangle(0, 0, 40, global::SCREEN_HEIGHT, sidebarColor);
00182
00183     DrawRectangle(global::SCREEN_WIDTH - 40, 0, 40, global::SCREEN_HEIGHT,
00184                 sidebarColor);
00185
00186     operationList.Draw();
00187     navigation.Draw();
00188
00189     animController->GetAnimation().Draw();
00190     codeHighlighter->Draw();
00191     footer.Draw(animController.get());
00192     DrawCurrentActionText();
00193     DrawCurrentErrorText();
00194 }
00195
00196 template< typename T >
00197 bool State::LLState< T >::Update(float dt) {
00198     animController->Update(dt);
00199     codeHighlighter->Highlight(
00200         animController->GetAnimation().GetHighlightedLine());
00201     codeHighlighter->AddActionDescription(
00202         animController->GetAnimation().GetActionDescription());
00203     return true;
00204 }
00205
00206 template< typename T >
00207 inline void State::LLState< T >::SetCurrentAction(std::string action) {
00208     mCurrentAction = action;
00209 }
00210
00211 template< typename T >
00212 inline void State::LLState< T >::SetCurrentError(std::string error) {
00213     mCurrentError = error;
00214 }
00215
00216 template< typename T >
00217 inline void State::LLState< T >::ClearError() {
00218     mCurrentError.clear();
00219 }
00220
00221 template< typename T >
00222 inline void State::LLState< T >::ClearAction() {
00223     mCurrentAction.clear();
00224 }
00225
00226 template< typename T >
00227 inline void State::LLState< T >::Success() {
00228     operationList.ToggleOperations();
00229     SetMouseCursor(MOUSE_CURSOR_DEFAULT);
```

```
00230     ClearError();
00231 }
00232
00233 template< typename T >
00234 inline void State::LLState< T >::DrawCurrentActionText() {
00235     const Color textColor = Settings::getInstance().getColor(ColorTheme::Text);
00236
00237     float rightAlignment = global::SCREEN_WIDTH - 50;
00238     Font font = GetContext().fonts->Get(FONT::Default_Bold);
00239     float width = MeasureTextEx(font, mCurrentAction.c_str(), 32, 0).x;
00240     float x = rightAlignment - width;
00241
00242     DrawTextEx(font, mCurrentAction.c_str(),
00243                 (Vector2){x, global::SCREEN_HEIGHT - 376}, 32, 0, textColor);
00244
00245     // codeHighlighter->SetPosition(), global::SCREEN_HEIGHT - 334);
00246 }
00247
00248 template< typename T >
00249 inline void State::LLState< T >::DrawCurrentErrorText() {
00250     const Color errorTextColor =
00251         Settings::getInstance().getColor(ColorTheme::Visualizer_ErrorText);
00252     Font font = GetContext().fonts->Get(FONT::Default_Bold);
00253     float width = MeasureTextEx(font, mCurrentError.c_str(), 24, 0).x;
00254     float x = 50;
00255
00256     DrawTextEx(
00257         font, mCurrentError.c_str(),
00258         (Vector2){x, global::SCREEN_HEIGHT - operationList.GetSize().y - 90},
00259         24, 0, errorTextColor);
00260 }
00261
00262 template< typename T >
00263 void State::LLState< T >::InitNavigationBar() {
00264     navigation.SetVisibleTitle(true);
00265     auto info = GetContext().categories->Get(CATEGORY::LinkedList);
00266     navigation.SetCategory(info.categoryName);
00267     navigation.SetActiveTitle(activeDS);
00268     for (auto dsID : info.mDS) {
00269         auto dsInfo = GetContext().dsInfo->Get(dsID);
00270         navigation.InsertTitle(dsID, dsInfo.stateID, dsInfo.abbr, dsInfo.name);
00271     };
00272 }
00273
00274 template< typename T >
00275 void State::LLState< T >::AddOperations() {
00276     AddInitializeOperation();
00277     AddInsertOperation();
00278     AddDeleteOperation();
00279     AddUpdateOperation();
00280     AddSearchOperation();
00281
00282     operationList.setPosition(
00283         0, global::SCREEN_HEIGHT - 60 - operationList.GetSize().y);
00284     operationList.InitActionBar();
00285 }
00286
00287 template< typename T >
00288 void State::LLState< T >::AddInitializeOperation() {}
00289
00290 template< typename T >
00291 void State::LLState< T >::AddInsertOperation() {}
00292
00293 template< typename T >
00294 void State::LLState< T >::AddDeleteOperation() {}
00295
00296 template< typename T >
00297 void State::LLState< T >::AddUpdateOperation() {}
00298
00299 template< typename T >
00300 void State::LLState< T >::AddSearchOperation() {}
00301
00302 template< typename T >
00303 void State::LLState< T >::AddNoFieldOperationOption(
00304     GUIComponent::OperationContainer::Ptr container, std::string title,
00305     std::function< void() > action) {
00306     GUIComponent::OptionInputField::Ptr button(
00307         new GUIComponent::OptionInputField(GetContext().fonts));
00308
00309     button.get()->SetNoFieldOption(title, action);
00310
00311     container.get()->pack(button);
00312 }
00313
00314 template< typename T >
00315 void State::LLState< T >::AddIntFieldOperationOption(
00316     GUIComponent::OperationContainer::Ptr container, std::string title,
```

```
00317 Core::Deque< IntegerInput > fields,
00318 std::function< void(std::map< std::string, std::string >) > action) {
00319 GUIComponent::OptionInputField::Ptr button(
00320     new GUIComponent::OptionInputField(getContext().fonts));
00321 Core::Deque< GUIComponent::InputField::Ptr > intFields;
00322 for (auto field : fields) {
00323     GUIComponent::IntegerInputField::Ptr intField(
00324         new GUIComponent::IntegerInputField(getContext().fonts));
00325     intField.get()>>SetLabel(field.label);
00326     intField.get()>>SetInputFieldSize((Vector2){(float)field.width, 30});
00327     intField.get()>>SetConstraint(field.minLength, field.maxLength);
00328     intField.get()>>Randomize();
00329     intFields.push_back(intField);
00330 }
00331
00332 button.get()>>SetOption(title, intFields, action);
00333
00334 container.get()>>pack(button);
00335 }
00336
00337 template< typename T >
00338 void State::LLState< T >::AddStringFieldOption(
00339     GUIComponent::OperationContainer::Ptr container, std::string title,
00340     std::string label,
00341     std::function< void(std::map< std::string, std::string >) > action) {
00342 GUIComponent::OptionInputField::Ptr button(
00343     new GUIComponent::OptionInputField(getContext().fonts));
00344 GUIComponent::StringInputField::Ptr strField(
00345     new GUIComponent::StringInputField(getContext().fonts));
00346 strField.get()>>SetLabel(label);
00347 strField.get()>>SetInputFieldSize((Vector2){100, 30});
00348 button.get()>>SetOption(title, {strField}, action);
00349 container.get()>>pack(button);
00350 }
00351
00352 #endif // STATES_LINKEDLIST_LLSTATE_HPP
```

## 8.149 src/States/LinkedList/QueueState.cpp File Reference

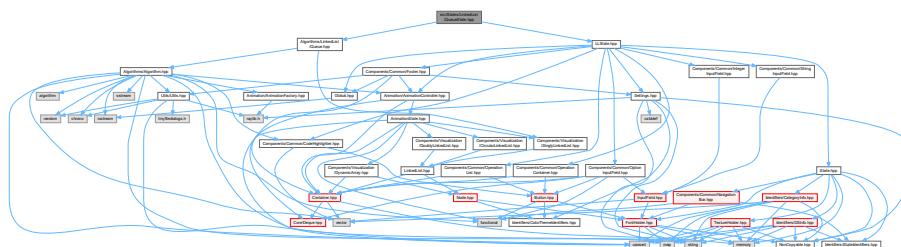
```
#include "QueueState.hpp"
#include <iostream>
#include "Global.hpp"
Include dependency graph for QueueState.cpp:
```



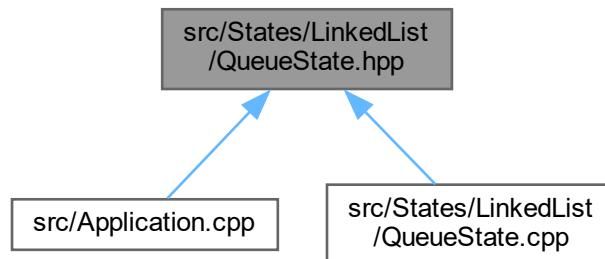
## 8.150 src/States/LinkedList/QueueState.hpp File Reference

```
#include "Algorithms/LinkedList/Queue.hpp"
#include "LLState.hpp"
```

Include dependency graph for QueueState.hpp:



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



## Classes

- class [State::QueueState](#)  
*The state that is used to visualize the queue.*

## Namespaces

- namespace [State](#)

## 8.151 QueueState.hpp

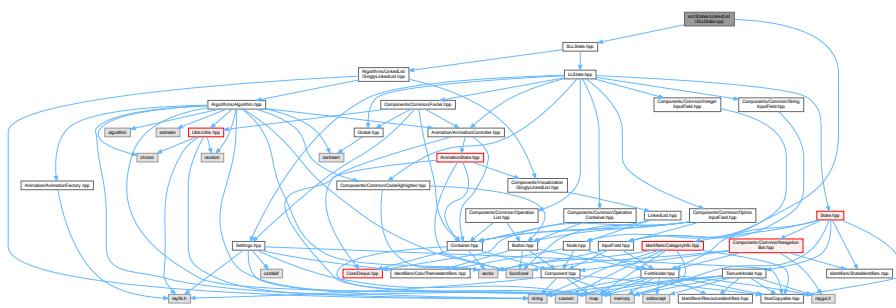
[Go to the documentation of this file.](#)

```
00001 #ifndef STATES_LINKEDLIST_QUEUESTATE_HPP
00002 #define STATES_LINKEDLIST_QUEUESTATE_HPP
00003
00004 #include "Algorithms/LinkedList/Queue.hpp"
00005 #include "LLState.hpp"
00006
00007 namespace State {
00013     class QueueState : public LLState< SLLAnimationController > {
00014     private:
00018         Algorithm::Queue queue;
00019
00020     public:
00024         void AddInsertOperation();
```

```
00025
00029     void AddInitializeOperation();
00030
00034     void AddDeleteOperation();
00035
00039     void AddSearchOperation();
00040
00041     public:
00042         QueueState(StateStack& stack, Context context);
00043
00044         ~QueueState();
00045     };
00046     // namespace State
00047
00048 #endif // STATES_LINKEDLIST_QUEUESTATE_HPP
```

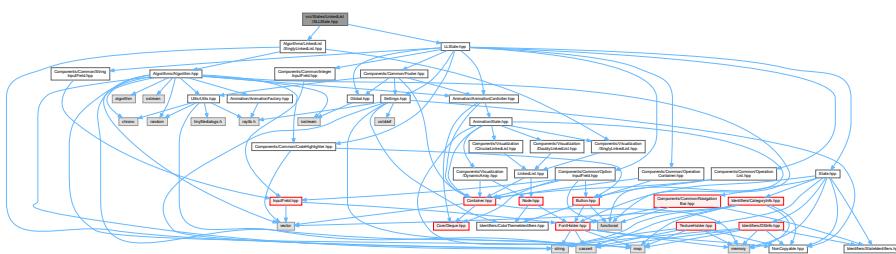
## 8.152 src/States/LinkedList/SLLState.cpp File Reference

```
#include "SLLState.hpp"
#include "Components/Visualization/Node.hpp"
Include dependency graph for SLLState.cpp:
```

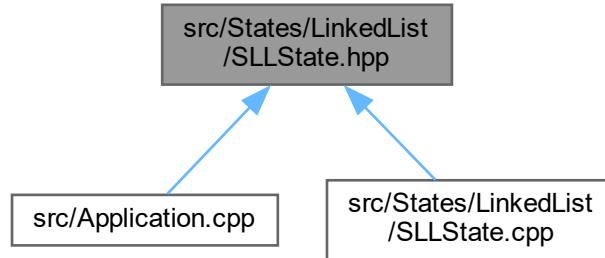


## 8.153 src/States/LinkedList/SLLState.hpp File Reference

```
#include "Algorithms/LinkedList/SinglyLinkedList.hpp"
#include "LLState.hpp"
Include dependency graph for SLLState.hpp:
```



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



## Classes

- class [State::SLLState](#)  
*The state class that is used to represent the singly linked list state/scene of the application.*

## Namespaces

- namespace [State](#)

## 8.154 SLLState.hpp

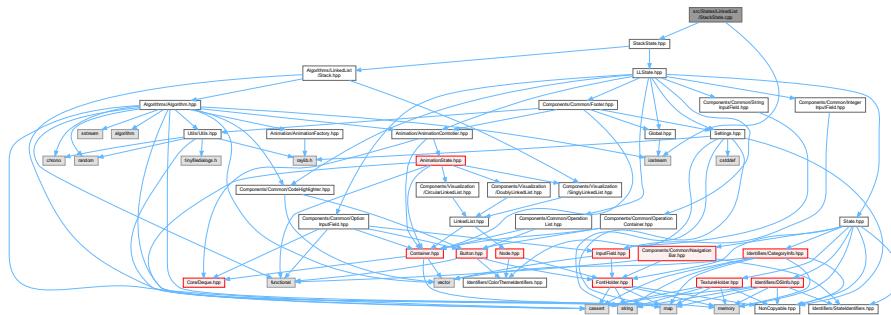
[Go to the documentation of this file.](#)

```

00001 #ifndef STATES_LINKEDLIST_SLLSTATE_HPP
00002 #define STATES_LINKEDLIST_SLLSTATE_HPP
00003
00004 #include "Algorithms/LinkedList/SinglyLinkedList.hpp"
00005 #include "LLState.hpp"
00006
00007 namespace State {
00012     class SLLState : public LLState< SLLAnimationController > {
00013     private:
00017         Algorithm::SinglyLinkedList SLL;
00018
00019     private:
00023         void AddInsertOperation();
00024
00029         void AddInitializeOperation();
00030
00034         void AddUpdateOperation();
00035
00039         void AddDeleteOperation();
00040
00044         void AddSearchOperation();
00045
00046     public:
00053         SLLState(StateStack& stack, Context context);
00054
00058         ~SLLState();
00059     };
00060 }; // namespace State
00061
00062 #endif // STATES_LINKEDLIST_SLLSTATE_HPP
  
```

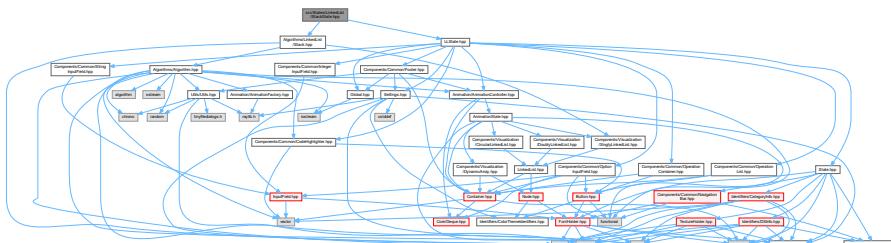
## 8.155 src/States/LinkedList/StackState.cpp File Reference

```
#include "StackState.hpp"
#include <iostream>
Include dependency graph for StackState.cpp:
```

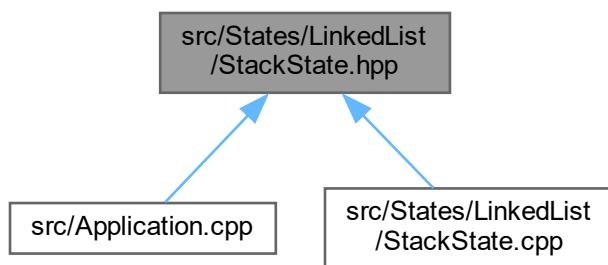


## 8.156 src/States/LinkedList/StackState.hpp File Reference

```
#include "Algorithms/LinkedList/Stack.hpp"
#include "LLState.hpp"
Include dependency graph for StackState.hpp:
```



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



## Classes

- class **State::StackState**

*The state class that is used to represent the stack state/scene of the application.*

## Namespaces

- namespace **State**

## 8.157 StackState.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef STATES_LINKEDLIST_STACKSTATE_HPP
00002 #define STATES_LINKEDLIST_STACKSTATE_HPP
00003
00004 #include "Algorithms/LinkedList/Stack.hpp"
00005 #include "LLState.hpp"
00006
00007 namespace State {
00014     class StackState : public LLState< SLLAnimationController > {
00015     private:
00019         Algorithm::Stack mStackAlgorithm;
00020
00021     public:
00025         void AddInsertOperation();
00026
00030         void AddInitializeOperation();
00031
00035         void AddDeleteOperation();
00036
00040         void AddSearchOperation();
00041
00042     public:
00048         StackState(StateStack& stack, Context context);
00049
00053         ~StackState();
00054     };
00055 }; // namespace State
00056
00057 #endif // STATES_LINKEDLIST_STACKSTATE_HPP
```

## 8.158 src/StateStack.cpp File Reference

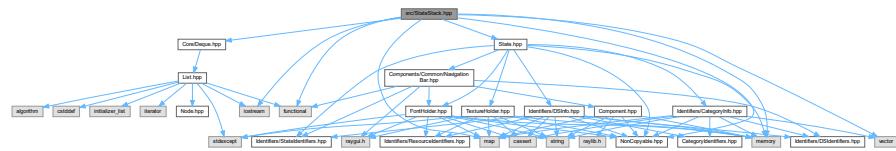
```
#include "StateStack.hpp"
#include <cassert>
#include <iostream>
Include dependency graph for StateStack.cpp:
```



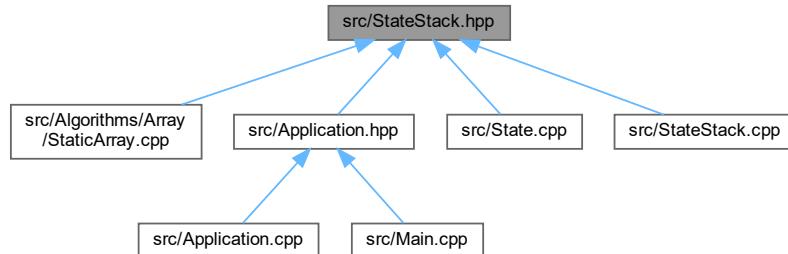
## 8.159 src/StateStack.hpp File Reference

```
#include <functional>
#include <iostream>
#include <map>
#include <memory>
#include <vector>
#include "Core/Deque.hpp"
#include "NonCopyable.hpp"
#include "State.hpp"
```

Include dependency graph for StateStack.hpp:



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



## Classes

- class [State::StateStack](#)  
*The state stack that is used to store the states (scenes) of the application and support the navigation between states (scenes).*
- struct [State::StateStack::PendingChange](#)  
*The pending changes that are applied to the state stack.*

## Namespaces

- namespace [State](#)

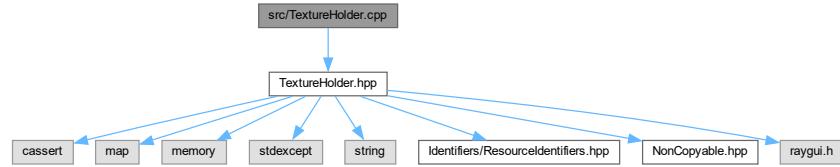
## 8.160 StateStack.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef STATESTACK_HPP
00002 #define STATESTACK_HPP
00003
00004 #include <functional>
00005 #include <iostream>
00006 #include <map>
00007 #include <memory>
00008 #include <vector>
00009
00010 #include "Core/Deque.hpp"
00011 #include "NonCopyable.hpp"
00012 #include "State.hpp"
00013
00014 namespace State {
00015     class StateStack : private NonCopyable< StateStack > {
00016     public:
00017         enum class Action {
00018             Push,
00019             Pop,
00020             Clear,
00021         };
00022
00023     public:
00024         explicit StateStack(State::Context context);
00025
00026         template< class T >
00027         void RegisterState(States::ID stateID);
00028
00029         void Update(float dt);
00030
00031         void Draw();
00032
00033         void PushState(States::ID stateID);
00034
00035         void PopState();
00036
00037         void ClearStates();
00038
00039         bool IsEmpty() const;
00040
00041     private:
00042         State::Ptr createState(States::ID stateID);
00043
00044         void ApplyPendingChanges();
00045
00046     private:
00047         struct PendingChange {
00048             explicit PendingChange(Action action,
00049                                     States::ID stateID = States::None);
00050             PendingChange();
00051             Action action;
00052             States::ID stateID;
00053         };
00054
00055     private:
00056         std::vector< State::Ptr > mStack;
00057
00058         Core::Deque< PendingChange > mPendingList;
00059
00060         State::Context mContext;
00061
00062         std::map< States::ID, std::function< State::Ptr() > > mFactories;
00063     };
00064 }; // namespace State
00065
00066 template< class T >
00067 void State::StateStack::RegisterState(States::ID stateID) {
00068     mFactories[stateID] = [this]() {
00069         return State::Ptr(new T(*this, mContext));
00070     };
00071 }
00072
00073 #endif // STATESTACK_HPP
```

## 8.161 src/TextureHolder.cpp File Reference

```
#include "TextureHolder.hpp"  
Include dependency graph for TextureHolder.cpp:
```



## 8.162 src/TextureHolder.hpp File Reference

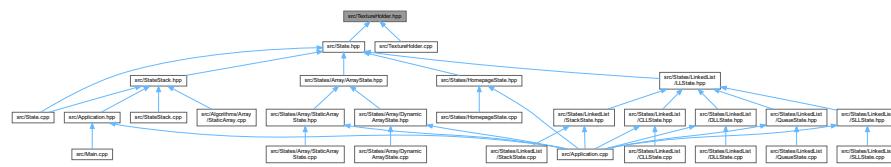
```
#include <cassert>
#include <map>
#include <memory>
#include <stdexcept>
#include <string>
#include "Identifiers/ResourceIdentifiers.hpp"
#include "NonCopyable.hpp"
#include "raygui.h"

Include dependency graph for TextureHolder.hpp:
```

Include dependency graph for `TextureHolder.hpp`:



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



## Classes

- class `TextureHolder`

*The image holder class that is used to store the images.*

## 8.163 TextureHolder.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef IMAGEHOLDER_HPP
00002 #define IMAGEHOLDER_HPP
00003
00004 #include <cassert>
00005 #include <map>
00006 #include <memory>
00007 #include <stdexcept>
00008 #include <string>
00009
00010 #include "Identifiers/ResourceIdentifiers.hpp"
00011 #include "NonCopyable.hpp"
00012 #include "raygui.h"
00013
00021 class TextureHolder : private NonCopyable< TextureHolder > {
00022 public:
00027     TextureHolder();
00028
00033     ~TextureHolder();
00034
00041     void Load(Textures::ID id, const std::string& filename);
00042
00054     void Load(Textures::ID id, const std::string& filename, int width,
00055             int height, bool cropCenter = false);
00056
00064     void LoadFromImage(Textures::ID id, Image& image);
00065
00077     void LoadFromImage(Textures::ID id, Image& image, int width, int height);
00078
00086     Texture& Get(Textures::ID id);
00087
00095     const Texture& Get(Textures::ID id) const;
00096
00097 private:
00105     void InsertResource(Textures::ID id, std::unique_ptr< Texture > texture);
00106
00107 private:
00114     std::map< Textures::ID, std::unique_ptr< Texture > > mTextureMap;
00115 };
00117 #endif // IMAGEHOLDER_HPP

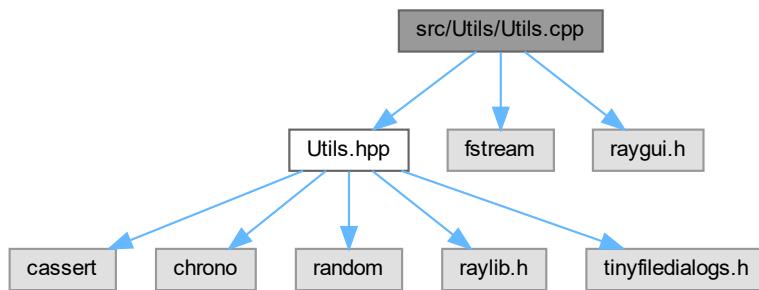
```

## 8.164 src/Utils/Utils.cpp File Reference

```

#include "Utils.hpp"
#include <fstream>
#include "raygui.h"
Include dependency graph for Utils.cpp:

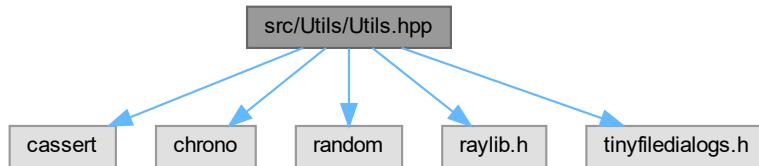
```



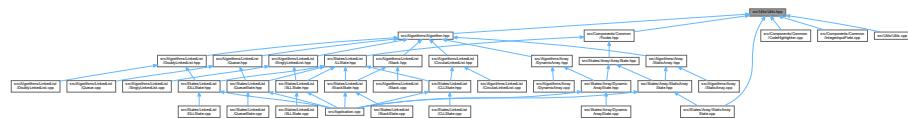
## 8.165 src/Utils/Utils.hpp File Reference

```
#include <cassert>
#include <chrono>
#include <random>
#include "raylib.h"
#include "tinyfiledialogs.h"
```

Include dependency graph for Utils.hpp:



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



## Namespaces

- namespace [Utils](#)

## Functions

- std::string [Utils::OpenFileDialog](#) (std::string title, std::string description, std::vector< std::string > filters, std::string defaultPath, bool allowMultipleSelect)
- std::string [Utils::ReadInputFromFile](#) (std::string path)
- int [Utils::Rand](#) (int lower, int upper)
- void [Utils::DrawTextBoxed](#) (Font font, const char \*text, Rectangle rec, float fontSize, float spacing, bool wordWrap, Color tint)
- void [Utils::DrawTextBoxedSelectable](#) (Font font, const char \*text, Rectangle rec, float fontSize, float spacing, bool wordWrap, Color tint, int selectStart, int selectLength, Color selectTint, Color selectBackTint)
- bool [Utils::DrawIcon](#) (int iconID, int x, int y, int pixelSize, Color color, Color hoverColor)

## 8.166 Utils.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef UTILS_HPP
00002 #define UTILS_HPP
00003
00004 #include <cassert>
00005 #include <chrono>
00006 #include <random>
00007
00008 #include "raylib.h"
00009 #include "tinyfiledialogs.h"
00010
00011 namespace Utils {
00012     std::string OpenFileDialog(std::string title, std::string description,
00013                               std::vector< std::string > filters,
00014                               std::string defaultPath,
00015                               bool allowMultipleSelect);
00016
00017     std::string ReadInputFromFile(std::string path);
00018
00019     // Returns a random integer in the range [lower, upper]
00020     int Rand(int lower, int upper);
00021
00022     // Draw text using font inside rectangle limits
00023     void DrawTextBoxed(Font font, const char* text, Rectangle rec,
00024                         float fontSize, float spacing, bool wordWrap,
00025                         Color tint);
00026
00027     // Draw text using font inside rectangle limits with support for text
00028     // selection;
00029     void DrawTextBoxedSelectable(Font font, const char* text, Rectangle rec,
00030                                 float fontSize, float spacing, bool wordWrap,
00031                                 Color tint, int selectStart, int selectLength,
00032                                 Color selectTint, Color selectBackTint);
00033     bool DrawIcon(int iconID, int x, int y, int pixelSize, Color color,
00034                   Color hoverColor);
00035 }; // namespace Utils
00036
00037 #endif // UTILS_HPP
```

# Index

~Algorithm  
    Algorithm::Algorithm< GUIAlgorithm, AnimationState >, [31](#)  
~AnimationController  
    Animation::AnimationController< T >, [39](#)  
~AnimationState  
    Animation::AnimationState< T >, [49](#)  
~Application  
    Application, [56](#)  
~ArrayState  
    State::ArrayState< T >, [63](#)  
~Button  
    GUIComponent::Button, [75](#)  
~CLLState  
    State::CLLState, [133](#)  
~Card  
    GUIComponent::Card, [89](#)  
~CircularLinkedList  
    Algorithm::CircularLinkedList, [102](#)  
    GUIVisualizer::CircularLinkedList, [116](#)  
~CodeHighlighter  
    GUIComponent::CodeHighlighter, [143](#)  
~Component  
    GUI::Component, [153](#)  
~DLLState  
    State::DLLState, [201](#)  
~DoublyLinkedList  
    Algorithm::DoublyLinkedList, [211](#)  
    GUIVisualizer::DoublyLinkedList, [226](#)  
~DynamicArray  
    Algorithm::DynamicArray, [245](#)  
    GUIVisualizer::DynamicArray, [258](#)  
~DynamicArrayState  
    State::DynamicArrayState, [275](#)  
~FontHolder  
    FontHolder, [283](#)  
~Footer  
    GUIComponent::Footer< T >, [289](#)  
~HomepageState  
    State::HomepageState, [300](#)  
~InputField  
    GUIComponent::InputField, [311](#)  
~IntegerInputField  
    GUIComponent::IntegerInputField, [325](#)  
~LLState  
    State::LLState< T >, [379](#)  
~LinkedList  
    GUIVisualizer::LinkedList, [346](#)  
~List  
    Core::List< T >, [361](#)  
~NavigationBar  
    GUIComponent::NavigationBar, [391](#)  
~Node  
    GUIVisualizer::Node, [408](#)  
~NonCopyable  
    NonCopyable< T >, [421](#)  
~OperationContainer  
    GUIComponent::OperationContainer, [425](#)  
~OperationList  
    GUIComponent::OperationList, [436](#)  
~OptionInputField  
    GUIComponent::OptionInputField, [448](#)  
~Queue  
    Algorithm::Queue, [461](#)  
~QueueState  
    State::QueueState, [490](#)  
~SLLState  
    State::SLLState, [535](#)  
~Settings  
    Settings, [499](#)  
~SinglyLinkedList  
    Algorithm::SinglyLinkedList, [505](#)  
    GUIVisualizer::SinglyLinkedList, [519](#)  
~Stack  
    Algorithm::Stack, [545](#)  
~StackState  
    State::StackState, [575](#)  
~State  
    State::State, [584](#)  
~StaticArray  
    Algorithm::StaticArray, [597](#)  
~StaticArrayState  
    State::StaticArrayState, [608](#)  
~StringInputField  
    GUIComponent::StringInputField, [618](#)  
~TextureHolder  
    TextureHolder, [629](#)  
abbr  
    DSInfo::Info, [307](#)  
abbrTitle  
    GUIComponent::NavigationBar::TitleInfo, [633](#)  
Access  
    Algorithm::DynamicArray, [245](#)  
    Algorithm::StaticArray, [597](#)  
Action  
    State::StateStack, [589](#)  
action  
    GUIComponent::Button, [83](#)

State::StateStack::PendingChange, 457  
 actionDescription  
     Animation::AnimationState< T >, 53  
 ActionDescription\_Background  
     ColorTheme, 20  
 ActionDescription\_Text  
     ColorTheme, 20  
 ActionList\_Background  
     ColorTheme, 20  
 ActionList\_HoverBackground  
     ColorTheme, 20  
 ActionList\_Text  
     ColorTheme, 20  
 Active  
     GUIVisualizer::CircularLinkedList, 116  
     GUIVisualizer::DoublyLinkedList, 226  
     GUIVisualizer::LinkedList, 345  
     GUIVisualizer::Node, 407  
     GUIVisualizer::SinglyLinkedList, 519  
 ActiveBlue  
     GUIVisualizer::Node, 407  
 activeDS  
     State::ArrayState< T >, 69  
     State::CLLState, 138  
     State::DLLState, 206  
     State::DynamicArrayList, 280  
     State::LLState< T >, 385  
     State::QueueState, 495  
     State::SLLState, 540  
     State::StackState, 580  
     State::StaticArrayList, 613  
 ActiveGreen  
     GUIVisualizer::Node, 407  
 ActiveRed  
     GUIVisualizer::Node, 407  
 activeTitle  
     GUIComponent::NavigationBar, 397  
 AddAccessOperation  
     State::ArrayState< T >, 63  
     State::DynamicArrayList, 275  
     State::StaticArrayList, 608  
 AddActionDescription  
     GUIComponent::CodeHighlighter, 143  
 AddAnimation  
     Animation::AnimationController< T >, 39  
 AddCode  
     GUIComponent::CodeHighlighter, 143  
 AddColor  
     GUIVisualizer::Node, 408  
 AddDeleteOperation  
     State::ArrayState< T >, 63  
     State::CLLState, 133  
     State::DLLState, 201  
     State::DynamicArrayList, 275  
     State::LLState< T >, 379  
     State::QueueState, 490  
     State::SLLState, 535  
     State::StackState, 575  
 AddInitializeOperation  
     State::ArrayState< T >, 64  
     State::CLLState, 133  
     State::DLLState, 201  
     State::DynamicArrayList, 275  
     State::LLState< T >, 379  
     State::QueueState, 490  
     State::SLLState, 535  
     State::StackState, 575  
     State::StaticArrayList, 608  
 AddInputField  
     GUIComponent::OptionInputField, 448  
 AddInsertOperation  
     State::ArrayState< T >, 64  
     State::CLLState, 133  
     State::DLLState, 201  
     State::DynamicArrayList, 275  
     State::LLState< T >, 379  
     State::QueueState, 490  
     State::SLLState, 535  
     State::StackState, 575  
     State::StaticArrayList, 608  
 AddIntFieldOperationOption  
     State::ArrayState< T >, 64  
     State::CLLState, 133  
     State::DLLState, 201  
     State::DynamicArrayList, 276  
     State::LLState< T >, 379  
     State::QueueState, 491  
     State::SLLState, 536  
     State::StackState, 576  
     State::StaticArrayList, 609  
 AddNoFieldOperationOption  
     State::ArrayState< T >, 64  
     State::CLLState, 134  
     State::DLLState, 202  
     State::DynamicArrayList, 276  
     State::LLState< T >, 380  
     State::QueueState, 491  
     State::SLLState, 536  
     State::StackState, 576  
     State::StaticArrayList, 609  
 AddOperation  
     GUIComponent::OperationList, 436  
 AddOperations  
     State::ArrayState< T >, 64  
     State::CLLState, 134  
     State::DLLState, 202  
     State::DynamicArrayList, 276  
     State::LLState< T >, 380  
     State::QueueState, 491  
     State::SLLState, 536  
     State::StackState, 576  
     State::StaticArrayList, 609  
 AddSearchOperation  
     State::ArrayState< T >, 65  
     State::CLLState, 134

State::DLLState, 202  
State::DynamicArrayState, 276  
State::LLState< T >, 380  
State::QueueState, 491  
State::SLLState, 536  
State::StackState, 576  
State::StaticArrayState, 609  
AddStringFieldOption  
  State::ArrayState< T >, 65  
  State::CLLState, 134  
  State::DLLState, 202  
  State::DynamicArrayState, 276  
  State::LLState< T >, 380  
  State::QueueState, 491  
  State::SLLState, 536  
  State::StackState, 576  
  State::StaticArrayState, 609  
AddSubmitField  
  GUIComponent::OptionInputField, 448  
AddUpdateOperation  
  State::ArrayState< T >, 65  
  State::CLLState, 134  
  State::DLLState, 202  
  State::DynamicArrayState, 277  
  State::LLState< T >, 381  
  State::QueueState, 492  
  State::SLLState, 537  
  State::StackState, 577  
  State::StaticArrayState, 610  
Algorithm, 15  
  Algorithm::Algorithm< GUIAlgorithm, AnimationState >, 30, 31  
Algorithm::Algorithm< GUIAlgorithm, AnimationState >, 29  
  ~Algorithm, 31  
  Algorithm, 30, 31  
  animController, 36  
  ApplyInput, 31  
  codeHighlighter, 36  
  Empty, 32  
  EmptyGenerator, 32  
  GenerateAnimation, 32  
  GenerateRelayoutAnimation, 33  
  InitAction, 33  
  Random, 33  
  RandomFixedSize, 33  
  RandomFixedSizeGenerator, 34  
  RandomGenerator, 34  
  ReadFromExternalFile, 34  
  ReadFromFileGenerator, 35  
  UserDefined, 35  
  UserDefinedGenerator, 35  
  visualizer, 36  
Algorithm::CircularLinkedList, 99  
  ~CircularLinkedList, 102  
  animController, 110  
  ApplyInput, 102  
  CircularLinkedList, 102  
  codeHighlighter, 110  
  DeleteHead, 103  
  DeleteMiddle, 103  
  DeleteTail, 103  
  Empty, 103  
  EmptyGenerator, 103  
  GenerateAnimation, 104  
  GenerateRelayoutAnimation, 104  
  HighlightArrowFromCur, 105  
  HighlightCircularArrow, 105  
  InitAction, 105  
  InsertAfterTail, 106  
  InsertHead, 106  
  InsertMiddle, 106  
  maxN, 110  
  Random, 107  
  RandomFixedSize, 107  
  RandomFixedSizeGenerator, 107  
  RandomGenerator, 107  
  ReadFromExternalFile, 108  
  ReadFromFileGenerator, 108  
  ResetVisualizer, 108  
  Search, 109  
  size, 109  
  Update, 109  
  UserDefined, 109  
  UserDefinedGenerator, 110  
  visualizer, 111  
Algorithm::DoublyLinkedList, 208  
  ~DoublyLinkedList, 211  
  animController, 220  
  ApplyInput, 211  
  codeHighlighter, 220  
  DeleteHead, 212  
  DeleteMiddle, 212  
  DeleteTail, 212  
  DoublyLinkedList, 211  
  Empty, 212  
  EmptyGenerator, 212  
  GenerateAnimation, 213  
  GenerateRelayoutAnimation, 213  
  HighlightArrowBoth, 214  
  HighlightArrowNext, 214  
  HighlightArrowPrev, 214  
  InitAction, 215  
  InsertAfterTail, 215  
  InsertHead, 215  
  InsertMiddle, 216  
  maxN, 220  
  Random, 216  
  RandomFixedSize, 216  
  RandomFixedSizeGenerator, 217  
  RandomGenerator, 217  
  ReadFromExternalFile, 217  
  ReadFromFileGenerator, 218  
  ResetVisualizer, 218  
  Search, 218  
  size, 218

Update, 219  
 UserDefined, 219  
 UserDefinedGenerator, 219  
 visualizer, 220  
**Algorithm::DynamicArray, 242**  
 ~DynamicArray, 245  
 Access, 245  
 animController, 252  
 ApplyInput, 245  
 codeHighlighter, 252  
 DynamicArray, 244  
 Empty, 246  
 EmptyGenerator, 246  
 GenerateAnimation, 246  
 GenerateRelayoutAnimation, 246  
 InitAction, 247  
 maxN, 252  
 PopBack, 247  
 Push, 247  
 PushBack, 248  
 Random, 248  
 RandomFixedSize, 248  
 RandomFixedSizeGenerator, 249  
 RandomGenerator, 249  
 ReadFromExternalFile, 249  
 ReadFromFileGenerator, 250  
 Remove, 250  
 ResetVisualizer, 250  
 Search, 250  
 size, 251  
 Update, 251  
 UserDefined, 251  
 UserDefinedGenerator, 252  
 visualizer, 253  
**Algorithm::Queue, 457**  
 ~Queue, 461  
 animController, 467  
 ApplyInput, 461  
 codeHighlighter, 467  
 Dequeue, 461  
 Empty, 461  
 EmptyGenerator, 461  
 Enqueue, 462  
 EnqueueEmpty, 462  
 GenerateAnimation, 462  
 GenerateRelayoutAnimation, 463  
 HighlightArrowFromCur, 463  
 InitAction, 463  
 maxN, 467  
 PeekBack, 464  
 PeekFront, 464  
 Queue, 460  
 Random, 464  
 RandomFixedSize, 464  
 RandomFixedSizeGenerator, 465  
 RandomGenerator, 465  
 ReadFromExternalFile, 465  
 ReadFromFileGenerator, 466  
 size, 466  
 UserDefined, 466  
 UserDefinedGenerator, 466  
 visualizer, 467  
**Algorithm::SinglyLinkedList, 502**  
 ~SinglyLinkedList, 505  
 animController, 513  
 ApplyInput, 505  
 codeHighlighter, 513  
 DeleteHead, 506  
 DeleteMiddle, 506  
 DeleteTail, 506  
 Empty, 506  
 EmptyGenerator, 506  
 GenerateAnimation, 507  
 GenerateRelayoutAnimation, 507  
 HighlightArrowFromCur, 508  
 InitAction, 508  
 InsertAfterTail, 508  
 InsertHead, 509  
 InsertMiddle, 509  
 maxN, 513  
 Random, 509  
 RandomFixedSize, 509  
 RandomFixedSizeGenerator, 510  
 RandomGenerator, 510  
 ReadFromExternalFile, 510  
 ReadFromFileGenerator, 511  
 ResetVisualizer, 511  
 Search, 511  
 SinglyLinkedList, 504, 505  
 size, 511  
 Update, 512  
 UserDefined, 512  
 UserDefinedGenerator, 512  
 visualizer, 513  
**Algorithm::Stack, 542**  
 ~Stack, 545  
 animController, 551  
 ApplyInput, 546  
 codeHighlighter, 551  
 Empty, 546  
 EmptyGenerator, 546  
 GenerateAnimation, 546  
 GenerateRelayoutAnimation, 547  
 HighlightArrowFromCur, 547  
 InitAction, 548  
 maxN, 552  
 mStackOrientation, 552  
 Peek, 548  
 Pop, 548  
 Push, 548  
 Random, 549  
 RandomFixedSize, 549  
 RandomFixedSizeGenerator, 549  
 RandomGenerator, 549  
 ReadFromExternalFile, 550  
 ReadFromFileGenerator, 550

size, 550  
Stack, 545  
UserDefined, 551  
UserDefinedGenerator, 551  
visualizer, 552  
Algorithm::StaticArray, 593  
  ~StaticArray, 597  
  Access, 597  
  animController, 603  
  ApplyInput, 597  
  codeHighlighter, 603  
  Delete, 597  
  Empty, 598  
  EmptyGenerator, 598  
  GenerateAnimation, 598  
  GenerateRelayoutAnimation, 599  
  InitAction, 599  
  Insert, 599  
  maxN, 604  
  Random, 600  
  RandomFixedSize, 600  
  RandomFixedSizeGenerator, 600  
  RandomGenerator, 601  
  ReadFromExternalFile, 601  
  ReadFromFileGenerator, 601  
  ResetVisualizer, 601  
  Search, 602  
  size, 602  
  StaticArray, 596  
  Update, 602  
  UserDefined, 603  
  UserDefinedGenerator, 603  
  visualizer, 604  
alignment  
  GUIComponent::Button, 83  
AlignmentCount  
  GUIComponent::Button, 75  
AllFieldDisableEdit  
  GUIComponent::InputField, 312  
  GUIComponent::IntegerInputField, 326  
  GUIComponent::StringInputField, 619  
animateNode  
  GUIVisualizer::Node, 417  
Animation, 15  
Animation::AnimationController< T >, 37  
  ~AnimationController, 39  
  AddAnimation, 39  
  AnimationController, 39  
  animationGroup, 46  
  Clear, 40  
  Continue, 40  
  CurrentAnimationIndex, 40  
  defaultSpeed, 46  
  Done, 40  
  GetAnimateFrame, 40  
  GetAnimation, 41  
  GetAnimationDuration, 41  
  GetAnimationIndex, 41  
  GetNumAnimation, 41  
  GetSpeed, 42  
  GetStopDuration, 42  
  InteractionAllow, 42  
  InteractionLock, 42  
  IsInteractionAllow, 43  
  IsPlaying, 43  
  mCurrentAnimationIndex, 46  
  mCurrStopDuration, 46  
  mInteractionLock, 46  
  mPlaying, 46  
  mSpeed, 47  
  Pause, 43  
  PopAnimation, 43  
  Ptr, 39  
  Reset, 43  
  ResetCurrent, 44  
  RunAll, 44  
  SetAnimation, 44  
  SetSpeed, 44  
  StepBackward, 45  
  StepForward, 45  
  stopDuration, 47  
  Update, 45  
Animation::AnimationState< T >, 47  
  ~AnimationState, 49  
  actionDescription, 53  
  AnimationState, 49  
  Done, 49  
  Draw, 49  
  GetActionDescription, 50  
  GetCurrentPlayingAt, 50  
  GetDataStructure, 50  
  GetDuration, 51  
  GetHighlightedLine, 51  
  mAnimation, 54  
  mCurrentPlayingAt, 54  
  mDataStructureBefore, 54  
  mDuration, 54  
  mHighlightedLine, 54  
  PlayingAt, 51  
  Reset, 51  
  SetActionDescription, 52  
  SetAnimation, 52  
  SetDuration, 52  
  SetHighlightLine, 53  
  SetSourceDataStructure, 53  
  Update, 53  
AnimationController  
  Animation::AnimationController< T >, 39  
AnimationController.hpp  
  CLLAnimationController, 666  
  DArrayAnimationController, 667  
  DLLAnimationController, 667  
  SLLAnimationController, 667  
AnimationFactory, 15  
  BlendColor, 16  
  BounceOut, 16

Dist, 16  
 DrawActiveArrow, 16  
 DrawCircularArrow, 17  
 DrawDirectionalArrow, 17  
 DrawDoubleActiveArrow, 17  
 DrawDoubleDirectionalArrow, 17  
 ElasticOut, 17  
 InverseVector, 18  
 MoveNode, 18  
 ReCalculateEnds, 18  
 animationGroup  
     Animation::AnimationController< T >, 46  
 AnimationOnNode  
     GUIVisualizer::Node, 408  
 AnimationState  
     Animation::AnimationState< T >, 49  
 AnimationState.hpp  
     CLLAnimation, 673  
     DArrayAnimation, 674  
     DLLAnimation, 674  
     SLLAnimation, 674  
 animController  
     Algorithm::Algorithm< GUIAlgorithm, Animation-  
         State >, 36  
     Algorithm::CircularLinkedList, 110  
     Algorithm::DoublyLinkedList, 220  
     Algorithm::DynamicArray, 252  
     Algorithm::Queue, 467  
     Algorithm::SinglyLinkedList, 513  
     Algorithm::Stack, 551  
     Algorithm::StaticArray, 603  
     State::ArrayState< T >, 69  
     State::CLLState, 138  
     State::DLLState, 206  
     State::DynamicArrayState, 280  
     State::LLState< T >, 385  
     State::QueueState, 495  
     State::SLLState, 540  
     State::StackState, 580  
     State::StaticArrayState, 613  
 Application, 55  
     ~Application, 56  
     Application, 56  
     categories, 58  
     Close, 56  
     closed, 58  
     dsInfo, 58  
     fonts, 59  
     images, 59  
     Init, 56  
     LoadResources, 57  
     mStack, 59  
     RegisterStates, 57  
     Render, 57  
     Run, 57  
     Update, 57  
     WindowClosed, 58  
 Application.cpp

RAYGUI\_IMPLEMENTATION, 677  
 ApplyInput  
     Algorithm::Algorithm< GUIAlgorithm, Animation-  
         State >, 31  
     Algorithm::CircularLinkedList, 102  
     Algorithm::DoublyLinkedList, 211  
     Algorithm::DynamicArray, 245  
     Algorithm::Queue, 461  
     Algorithm::SinglyLinkedList, 505  
     Algorithm::Stack, 546  
     Algorithm::StaticArray, 597  
 ApplyPendingChanges  
     State::StateStack, 590  
 Array  
     Category, 19  
 ArrayState  
     State::ArrayState< T >, 63  
 arrowState  
     GUIVisualizer::CircularLinkedList, 127  
     GUIVisualizer::SinglyLinkedList, 529  
 arrowStateNext  
     GUIVisualizer::DoublyLinkedList, 237  
 arrowStatePrev  
     GUIVisualizer::DoublyLinkedList, 237  
 ArrowType  
     CircularLinkedList.cpp, 646  
     DoublyLinkedList.cpp, 650  
     GUIVisualizer::CircularLinkedList, 115  
     GUIVisualizer::DoublyLinkedList, 225  
     GUIVisualizer::LinkedList, 345  
     GUIVisualizer::SinglyLinkedList, 518  
     Queue.cpp, 654  
     SinglyLinkedList.cpp, 658  
     Stack.hpp, 664  
 ArrowTypeCount  
     GUIVisualizer::CircularLinkedList, 116  
     GUIVisualizer::DoublyLinkedList, 226  
     GUIVisualizer::LinkedList, 345  
     GUIVisualizer::SinglyLinkedList, 519  
 assign  
     Core::Deque< T >, 183, 184  
     Core::List< T >, 362, 363  
     Core::Queue< T >, 472, 473  
     Core::Stack< T >, 556, 557  
 at  
     Core::Deque< T >, 184, 185  
     Core::List< T >, 363  
     Core::Queue< T >, 473, 474  
     Core::Stack< T >, 557, 558  
 back  
     Core::Deque< T >, 185  
     Core::List< T >, 364  
     Core::Queue< T >, 474  
     Core::Stack< T >, 558, 559  
 Background  
     ColorTheme, 20  
 begin  
     Core::Deque< T >, 186

Core::List< T >, 365  
Core::Queue< T >, 474, 475  
Core::Stack< T >, 559  
Blank  
    Textures, 26  
BlendColor  
    AnimationFactory, 16  
BounceOut  
    AnimationFactory, 16  
bound  
    GUIComponent::Button, 83  
Button  
    GUIComponent::Button, 75  
Button\_Background  
    ColorTheme, 20  
Button\_HoveredBackground  
    ColorTheme, 20  
Button\_Text  
    ColorTheme, 20  
buttonColorTheme  
    GUIComponent::Button, 83  
buttonHoverColorTheme  
    GUIComponent::Button, 83  
buttons  
    GUIComponent::OperationList, 443  
capacity  
    GUIVisualizer::DynamicArray, 270  
Card  
    GUIComponent::Card, 88  
Card\_Background  
    ColorTheme, 20  
Card\_Text  
    ColorTheme, 20  
categories  
    Application, 58  
    State::State::Context, 179  
Category, 18  
    Array, 19  
    Count, 19  
    ID, 18  
    LinkedList, 19  
    None, 19  
categoryID  
    CategoryInfo::Info, 305  
    DSInfo::Info, 307  
CategoryInfo, 96  
    Get, 97  
    mFactories, 98  
    Register, 98  
CategoryInfo::Info, 304  
    categoryID, 305  
    categoryName, 305  
    Info, 304  
    mDS, 305  
categoryName  
    CategoryInfo::Info, 305  
Center  
    GUIComponent::Button, 75  
Circle  
    GUIVisualizer::Node, 407  
circularArrowState  
    GUIVisualizer::CircularLinkedList, 127  
CircularLinkedList  
    Algorithm::CircularLinkedList, 102  
    DataStructures, 22  
    GUIVisualizer::CircularLinkedList, 116  
    States, 26  
    Textures, 26  
CircularLinkedList.cpp  
    ArrowType, 646  
Clear  
    Animation::AnimationController< T >, 40  
    GUIVisualizer::DynamicArray, 258  
    State::StateStack, 590  
clear  
    Core::Deque< T >, 186  
    Core::List< T >, 365  
    Core::Queue< T >, 475  
    Core::Stack< T >, 559  
ClearAction  
    State::ArrayList< T >, 65  
    State::CLLState, 135  
    State::DLLState, 203  
    State::DynamicArrayList, 277  
    State::LLState< T >, 381  
    State::QueueState, 492  
    State::SLLState, 537  
    State::StackState, 577  
    State::StaticArrayList, 610  
ClearError  
    State::ArrayList< T >, 66  
    State::CLLState, 135  
    State::DLLState, 203  
    State::DynamicArrayList, 277  
    State::LLState< T >, 381  
    State::QueueState, 492  
    State::SLLState, 537  
    State::StackState, 577  
    State::StaticArrayList, 610  
ClearLabel  
    GUIVisualizer::Node, 408  
ClearStates  
    State::StateStack, 590  
ClearTitle  
    GUIComponent::NavigationBar, 391  
CLL  
    State::CLLState, 138  
CLLAnimation  
    AnimationState.hpp, 673  
CLLAnimationController  
    AnimationController.hpp, 666  
CLLState  
    State::CLLState, 132  
Close  
    Application, 56  
closed

Application, 58  
 CodeHighlighter  
     GUIComponent::CodeHighlighter, 142  
 codeHighlighter  
     Algorithm::Algorithm< GUIAlgorithm, AnimationState >, 36  
     Algorithm::CircularLinkedList, 110  
     Algorithm::DoublyLinkedList, 220  
     Algorithm::DynamicArray, 252  
     Algorithm::Queue, 467  
     Algorithm::SinglyLinkedList, 513  
     Algorithm::Stack, 551  
     Algorithm::StaticArray, 603  
     State::ArrayList< T >, 69  
     State::CLLState, 138  
     State::DLLState, 206  
     State::DynamicArrayList, 280  
     State::LLState< T >, 386  
     State::QueueState, 495  
     State::SLLState, 540  
     State::StackState, 580  
     State::StaticArrayList, 613  
 CodeHighlighter\_Background  
     ColorTheme, 20  
 CodeHighlighter\_HighlightedLineBackground  
     ColorTheme, 20  
 CodeHighlighter\_Text  
     ColorTheme, 20  
 ColorTheme, 19  
     ActionDescription\_Background, 20  
     ActionDescription\_Text, 20  
     ActionList\_Background, 20  
     ActionList\_HoverBackground, 20  
     ActionList\_Text, 20  
     Background, 20  
     Button\_Background, 20  
     Button\_HoveredBackground, 20  
     Button\_Text, 20  
     Card\_Background, 20  
     Card\_Text, 20  
     CodeHighlighter\_Background, 20  
     CodeHighlighter\_HighlightedLineBackground, 20  
     CodeHighlighter\_Text, 20  
     Count, 21  
     Footer\_Background, 20  
     Footer\_HoveredIcon, 20  
     Footer\_Icon, 20  
     ID, 19  
     InputField\_Inactive, 20  
     Logo1FirstPart, 20  
     Logo1SecondPart, 20  
     Logo2FirstPart, 20  
     Logo2SecondPart, 20  
     NavigationBar\_Background, 20  
     NavigationBar\_SelectedTitle, 20  
     NavigationBar\_UnselectedTitle, 20  
     Text, 20  
     Visualizer\_ActionText, 20  
 Visualizer\_Arrow\_Active, 21  
 Visualizer\_Arrow\_Default, 21  
 Visualizer\_ErrorText, 20  
 Visualizer\_Label, 20  
 Visualizer\_Node\_Active\_Background1, 20  
 Visualizer\_Node\_Active\_Background2, 20  
 Visualizer\_Node\_Active\_Outline1, 20  
 Visualizer\_Node\_Active\_Outline2, 20  
 Visualizer\_Node\_Active\_Text1, 20  
 Visualizer\_Node\_Active\_Text2, 20  
 Visualizer\_Node\_ActiveBlue\_Background1, 20  
 Visualizer\_Node\_ActiveBlue\_Background2, 20  
 Visualizer\_Node\_ActiveBlue\_Outline1, 20  
 Visualizer\_Node\_ActiveBlue\_Outline2, 20  
 Visualizer\_Node\_ActiveBlue\_Text1, 20  
 Visualizer\_Node\_ActiveBlue\_Text2, 20  
 Visualizer\_Node\_ActiveGreen\_Background1, 21  
 Visualizer\_Node\_ActiveGreen\_Background2, 21  
 Visualizer\_Node\_ActiveGreen\_Outline1, 20  
 Visualizer\_Node\_ActiveGreen\_Outline2, 21  
 Visualizer\_Node\_ActiveGreen\_Text1, 21  
 Visualizer\_Node\_ActiveGreen\_Text2, 21  
 Visualizer\_Node\_ActiveRed\_Background1, 21  
 Visualizer\_Node\_ActiveRed\_Background2, 21  
 Visualizer\_Node\_ActiveRed\_Outline1, 21  
 Visualizer\_Node\_ActiveRed\_Outline2, 21  
 Visualizer\_Node\_ActiveRed\_Text1, 21  
 Visualizer\_Node\_ActiveRed\_Text2, 21  
 Visualizer\_Node\_Default\_Background1, 20  
 Visualizer\_Node\_Default\_Background2, 20  
 Visualizer\_Node\_Default\_Outline1, 20  
 Visualizer\_Node\_Default\_Outline2, 20  
 Visualizer\_Node\_Default\_Text1, 20  
 Visualizer\_Node\_Default\_Text2, 20  
 Visualizer\_Node\_Iterated\_Background1, 21  
 Visualizer\_Node\_Iterated\_Background2, 21  
 Visualizer\_Node\_Iterated\_Outline1, 21  
 Visualizer\_Node\_Iterated\_Outline2, 21  
 Visualizer\_Node\_Iterated\_Text1, 21  
 Visualizer\_Node\_Iterated\_Text2, 21  
 Component  
     GUI::Component, 153  
 Consolas  
     Fonts, 22  
 const\_iterator  
     Core::List< T >::const\_iterator, 162, 163  
 Container  
     GUI::Container, 171  
 content  
     GUIComponent::Button, 83  
     GUIComponent::StringInputField, 625  
 Context  
     State::State::Context, 178  
 Continue  
     Animation::AnimationController< T >, 40  
 Core, 21  
 Core::Deque< T >, 180  
     assign, 183, 184

at, 184, 185  
back, 185  
begin, 186  
clear, 186  
empty, 186  
end, 187  
front, 187  
get\_iterator, 188  
insert\_previous, 188  
List, 183, 188, 189  
mBegin, 196  
mEnd, 197  
move\_previous, 189  
mSize, 197  
operator=, 189, 190  
operator[], 190, 191  
pop\_back, 191  
pop\_front, 191  
push\_back, 192  
push\_front, 192  
remove, 192, 193  
remove\_if, 194  
reset, 194  
resize, 194  
reverse, 195  
size, 195  
swap, 195  
unique, 196

Core::List< T >, 358  
~List, 361  
assign, 362, 363  
at, 363  
back, 364  
begin, 365  
clear, 365  
empty, 365  
end, 365, 366  
front, 366  
get\_iterator, 366  
insert\_previous, 367  
List, 361, 362  
mBegin, 374  
mEnd, 374  
move\_previous, 367  
mSize, 374  
operator=, 367, 368  
operator[], 368, 369  
pop\_back, 369  
pop\_front, 369  
push\_back, 369  
push\_front, 370  
remove, 370, 371  
remove\_if, 371  
reset, 372  
resize, 372  
reverse, 372  
size, 373  
swap, 373

unique, 373  
Core::List< T >::const\_iterator, 160  
const\_iterator, 162, 163  
difference\_type, 162  
iterator\_category, 162  
List, 168  
operator!=, 164  
operator\*, 164  
operator+, 164  
operator++, 165  
operator-, 165  
operator->, 166  
operator--, 165, 166  
operator=, 166, 167  
operator==, 167  
pointer, 162  
ptr, 168  
reference, 162  
swap, 167  
value\_type, 162

Core::List< T >::iterator, 334  
difference\_type, 336  
iterator, 336, 337  
iterator\_category, 336  
List, 340  
operator!=, 337  
operator\*, 337  
operator+, 337  
operator++, 338  
operator-, 338  
operator->, 339  
operator--, 339  
operator=, 339  
operator==, 340  
pointer, 336  
ptr, 341  
reference, 336  
swap, 340  
value\_type, 336

Core::Node< T >, 399  
mNext, 401  
mPrev, 401  
mValue, 402  
Node, 400, 401

Core::Queue< T >, 468  
assign, 472, 473  
at, 473, 474  
back, 474  
begin, 474, 475  
clear, 475  
empty, 475  
end, 475  
front, 475, 476  
get\_iterator, 476  
insert\_previous, 476  
List, 472, 477, 478  
mBegin, 486  
mEnd, 486

move\_previous, 478  
 mSize, 486  
 operator=, 478, 479  
 operator[], 479  
 pop, 480  
 pop\_back, 480  
 pop\_front, 480  
 push, 480, 481  
 push\_back, 481  
 push\_front, 481  
 remove, 481, 482  
 remove\_if, 483  
 reset, 483  
 resize, 483, 484  
 reverse, 484  
 size, 484  
 swap, 484, 485  
 unique, 485  
**Core::Stack< T >**, 553  
 assign, 556, 557  
 at, 557, 558  
 back, 558, 559  
 begin, 559  
 clear, 559  
 empty, 559  
 end, 560  
 front, 560  
 get\_iterator, 561  
 insert\_previous, 561  
 List, 556, 562  
 mBegin, 571  
 mEnd, 571  
 move\_previous, 563  
 mSize, 571  
 operator=, 563  
 operator[], 564  
 pop, 565  
 pop\_back, 565  
 pop\_front, 565  
 push, 565  
 push\_back, 566  
 push\_front, 566  
 remove, 566, 567  
 remove\_if, 568  
 reset, 568  
 resize, 568  
 reverse, 569  
 size, 569  
 swap, 569, 570  
 top, 570  
 unique, 570  
**Count**  
 Category, 19  
 ColorTheme, 21  
 DataStructures, 22  
 Fonts, 22  
 States, 26  
 Textures, 26  
**Courier**  
 Fonts, 22  
**Courier\_Bold**  
 Fonts, 22  
**CreateCard**  
 State::HomepageState, 300  
**createState**  
 State::StateStack, 590  
**CurrentAnimationIndex**  
 Animation::AnimationController< T >, 40  
**currentCategory**  
 GUIComponent::NavigationBar, 397  
**DArrayAnimation**  
 AnimationState.hpp, 674  
**DArrayAnimationController**  
 AnimationController.hpp, 667  
**DataStructures**, 21  
 CircularLinkedList, 22  
 Count, 22  
 DoublyLinkedList, 22  
 DynamicArray, 22  
 ID, 22  
 None, 22  
 Queue, 22  
 SinglyLinkedList, 22  
 Stack, 22  
 StaticArray, 22  
**Default**  
 Fonts, 22  
 GUIVisualizer::CircularLinkedList, 116  
 GUIVisualizer::DoublyLinkedList, 226  
 GUIVisualizer::LinkedList, 345  
 GUIVisualizer::Node, 407  
 GUIVisualizer::SinglyLinkedList, 519  
**Default\_Bold**  
 Fonts, 22  
**Default\_Italic**  
 Fonts, 22  
**defaultSpeed**  
 Animation::AnimationController< T >, 46  
**Delete**  
 Algorithm::StaticArray, 597  
**DeleteHead**  
 Algorithm::CircularLinkedList, 103  
 Algorithm::DoublyLinkedList, 212  
 Algorithm::SinglyLinkedList, 506  
**DeleteMiddle**  
 Algorithm::CircularLinkedList, 103  
 Algorithm::DoublyLinkedList, 212  
 Algorithm::SinglyLinkedList, 506  
**DeleteNode**  
 GUIVisualizer::CircularLinkedList, 117  
 GUIVisualizer::DoublyLinkedList, 227  
 GUIVisualizer::DynamicArray, 258  
 GUIVisualizer::LinkedList, 347  
 GUIVisualizer::SinglyLinkedList, 520  
**DeleteTail**  
 Algorithm::CircularLinkedList, 103

Algorithm::DoublyLinkedList, 212  
Algorithm::SinglyLinkedList, 506  
Dequeue  
    Algorithm::Queue, 461  
deselect  
    GUI::Component, 154  
    GUI::Container, 171  
    GUIComponent::Button, 75  
    GUIComponent::Card, 89  
    GUIComponent::CodeHighlighter, 143  
    GUIComponent::Footer< T >, 290  
    GUIComponent::InputField, 312  
    GUIComponent::IntegerInputField, 326  
    GUIComponent::NavigationBar, 391  
    GUIComponent::OperationContainer, 425  
    GUIComponent::OperationList, 436  
    GUIComponent::OptionInputField, 448  
    GUIComponent::StringInputField, 619  
    GUIVisualizer::CircularLinkedList, 117, 118  
    GUIVisualizer::DoublyLinkedList, 227, 228  
    GUIVisualizer::DynamicArray, 259  
    GUIVisualizer::LinkedList, 347, 348  
    GUIVisualizer::Node, 409  
    GUIVisualizer::SinglyLinkedList, 520, 521  
State::ArrayState< T >, 66  
State::CLLState, 135  
State::DLLState, 203  
State::DynamicArrayList, 277  
State::HomepageState, 300  
State::LLState< T >, 381  
State::QueueState, 492  
State::SLLState, 537  
State::StackState, 577  
State::State, 584  
State::StateStack, 591  
State::StaticArrayList, 610  
DrawActionDescription  
    GUIComponent::CodeHighlighter, 144  
DrawActiveArrow  
    AnimationFactory, 16  
DrawArrow  
    GUIVisualizer::CircularLinkedList, 118  
    GUIVisualizer::DoublyLinkedList, 228  
    GUIVisualizer::SinglyLinkedList, 521  
DrawButtonText  
    GUIComponent::Button, 77  
DrawCircularArrow  
    AnimationFactory, 17  
DrawCodeHighlighter  
    GUIComponent::CodeHighlighter, 144  
DrawCurrent  
    GUI::Container, 172  
    GUIComponent::Footer< T >, 290  
    GUIComponent::OperationContainer, 426  
    GUIComponent::OperationList, 437  
    GUIComponent::OptionInputField, 449  
    GUIVisualizer::CircularLinkedList, 118  
    GUIVisualizer::DoublyLinkedList, 228  
    GUIVisualizer::DynamicArray, 259  
    GUIVisualizer::LinkedList, 348  
    GUIVisualizer::SinglyLinkedList, 521  
DrawCurrentActionText  
    State::ArrayState< T >, 66  
    State::CLLState, 135  
    State::DLLState, 203  
    State::DynamicArrayList, 277  
    State::LLState< T >, 381  
    State::QueueState, 492  
    State::SLLState, 537  
    State::StackState, 577  
    State::StaticArrayList, 610  
Done  
    Animation::AnimationController< T >, 40  
    Animation::AnimationState< T >, 49  
DoublyLinkedList  
    Algorithm::DoublyLinkedList, 211  
    DataStructures, 22  
    GUIVisualizer::DoublyLinkedList, 226  
    States, 26  
    Textures, 26  
DoublyLinkedList.cpp  
    ArrowType, 650  
Draw  
    Animation::AnimationState< T >, 49  
    GUI::Component, 154  
    GUI::Container, 171  
    GUIComponent::Button, 76  
    GUIComponent::Card, 89  
    GUIComponent::CodeHighlighter, 143  
    GUIComponent::Footer< T >, 290  
    GUIComponent::InputField, 312  
    GUIComponent::IntegerInputField, 326  
    GUIComponent::NavigationBar, 391  
    GUIComponent::OperationContainer, 425  
    GUIComponent::OperationList, 436  
    GUIComponent::OptionInputField, 448  
    GUIComponent::StringInputField, 619  
    GUIVisualizer::CircularLinkedList, 117, 118  
    GUIVisualizer::DoublyLinkedList, 227, 228  
    GUIVisualizer::DynamicArray, 259  
    GUIVisualizer::LinkedList, 347, 348  
    GUIVisualizer::Node, 409  
    GUIVisualizer::SinglyLinkedList, 520, 521

DrawCurrentErrorText  
 State::ArrayState< T >, 66  
 State::CLLState, 135  
 State::DLLState, 203  
 State::DynamicArrayState, 278  
 State::LLState< T >, 382  
 State::QueueState, 493  
 State::SLLState, 538  
 State::StackState, 578  
 State::StaticArrayState, 611

DrawDirectionalArrow  
 AnimationFactory, 17

DrawDoubleActiveArrow  
 AnimationFactory, 17

DrawDoubleDirectionalArrow  
 AnimationFactory, 17

DrawField  
 GUIComponent::InputField, 312  
 GUIComponent::IntegerInputField, 326  
 GUIComponent::StringInputField, 619

DrawIcon  
 Utils, 27

DrawImage  
 GUIComponent::Card, 89

DrawIntroduction  
 State::HomepageState, 301

DrawLabel  
 GUIVisualizer::Node, 409

DrawLogo  
 GUIComponent::NavigationBar, 392

DrawNode  
 GUIVisualizer::Node, 409

DrawSwitchTheme  
 GUIComponent::NavigationBar, 392

DrawTextBoxed  
 Utils, 27

DrawTextBoxedSelectable  
 Utils, 27

DrawTitle  
 GUIComponent::Card, 89

DrawTitles  
 GUIComponent::NavigationBar, 392

DSInfo, 239  
 Get, 240  
 mFactories, 241  
 Register, 241

dslInfo  
 Application, 58  
 State::State::Context, 179

DSInfo::Info, 306  
 abbr, 307  
 categoryID, 307  
 ID, 307  
 Info, 306  
 name, 307  
 stateID, 308  
 thumbnail, 308

DynamicArray  
 Algorithm::DynamicArray, 244  
 DataStructures, 22  
 GUIVisualizer::DynamicArray, 257  
 States, 26  
 Textures, 26

DynamicArrayList  
 State::DynamicArrayState, 274

editMode  
 GUIComponent::InputField, 318  
 GUIComponent::IntegerInputField, 332  
 GUIComponent::StringInputField, 625

ElasticOut  
 AnimationFactory, 17

Empty  
 Algorithm::Algorithm< GUIAlgorithm, AnimationState >, 32  
 Algorithm::CircularLinkedList, 103  
 Algorithm::DoublyLinkedList, 212  
 Algorithm::DynamicArray, 246  
 Algorithm::Queue, 461  
 Algorithm::SinglyLinkedList, 506  
 Algorithm::Stack, 546  
 Algorithm::StaticArray, 598

empty  
 Core::Deque< T >, 186  
 Core::List< T >, 365  
 Core::Queue< T >, 475  
 Core::Stack< T >, 559

EmptyGenerator  
 Algorithm::Algorithm< GUIAlgorithm, AnimationState >, 32  
 Algorithm::CircularLinkedList, 103  
 Algorithm::DoublyLinkedList, 212  
 Algorithm::DynamicArray, 246  
 Algorithm::Queue, 461  
 Algorithm::SinglyLinkedList, 506  
 Algorithm::Stack, 546  
 Algorithm::StaticArray, 598

EnableFitContent  
 GUIComponent::Button, 77

end  
 Core::Deque< T >, 187  
 Core::List< T >, 365, 366  
 Core::Queue< T >, 475  
 Core::Stack< T >, 560

Enqueue  
 Algorithm::Queue, 462

EnqueueEmpty  
 Algorithm::Queue, 462

extractedValue  
 GUIComponent::InputField, 318  
 GUIComponent::IntegerInputField, 332  
 GUIComponent::StringInputField, 625

ExtractInput  
 GUIComponent::OptionInputField, 449

ExtractValue  
 GUIComponent::InputField, 313  
 GUIComponent::IntegerInputField, 327

GUIComponent::StringInputField, 620  
Favicon  
    Textures, 26  
favicon  
    global, 23  
fields  
    GUIComponent::InputField, 318  
    GUIComponent::IntegerInputField, 332  
    GUIComponent::StringInputField, 626  
FitContent  
    GUIComponent::Button, 77  
fitContent  
    GUIComponent::Button, 84  
FontHolder, 282  
    ~FontHolder, 283  
    FontHolder, 283  
    Get, 284  
    InsertResource, 285  
    Load, 285  
    mFontMap, 286  
Fonts, 22  
    Consolas, 22  
    Count, 22  
    Courier, 22  
    Courier\_Bold, 22  
    Default, 22  
    Default\_Bold, 22  
    Default\_Italic, 22  
    ID, 22  
    Silkscreen, 22  
fonts  
    Application, 59  
    GUIComponent::Button, 84  
    GUIComponent::Card, 94  
    GUIComponent::CodeHighlighter, 149  
    GUIComponent::InputField, 319  
    GUIComponent::IntegerInputField, 333  
    GUIComponent::NavigationBar, 398  
    GUIComponent::OptionInputField, 454  
    GUIComponent::StringInputField, 626  
    GUIVisualizer::CircularLinkedList, 127  
    GUIVisualizer::DoublyLinkedList, 237  
    GUIVisualizer::DynamicArray, 270  
    GUIVisualizer::LinkedList, 355  
    GUIVisualizer::Node, 417  
    GUIVisualizer::SinglyLinkedList, 530  
    State::State::Context, 179  
fontSize  
    GUIComponent::Button, 84  
Footer  
    GUIComponent::Footer< T >, 289  
footer  
    State::ArrayState< T >, 69  
    State::CLLState, 138  
    State::DLLState, 206  
    State::DynamicArrayState, 281  
    State::LLState< T >, 386  
    State::QueueState, 496  
    State::SLLState, 541  
    State::StackState, 581  
    State::StaticArrayList, 614  
Footer\_Background  
    ColorTheme, 20  
Footer\_HoveredIcon  
    ColorTheme, 20  
Footer\_Icon  
    ColorTheme, 20  
front  
    Core::Deque< T >, 187  
    Core::List< T >, 366  
    Core::Queue< T >, 475, 476  
    Core::Stack< T >, 560  
GenerateAnimation  
    Algorithm::Algorithm< GUIAlgorithm, AnimationState >, 32  
    Algorithm::CircularLinkedList, 104  
    Algorithm::DoublyLinkedList, 213  
    Algorithm::DynamicArray, 246  
    Algorithm::Queue, 462  
    Algorithm::SinglyLinkedList, 507  
    Algorithm::Stack, 546  
    Algorithm::StaticArrayList, 598  
GenerateNode  
    GUIVisualizer::CircularLinkedList, 118  
    GUIVisualizer::DoublyLinkedList, 228  
    GUIVisualizer::DynamicArray, 259  
    GUIVisualizer::LinkedList, 348  
    GUIVisualizer::SinglyLinkedList, 521  
GenerateRelayoutAnimation  
    Algorithm::Algorithm< GUIAlgorithm, AnimationState >, 33  
    Algorithm::CircularLinkedList, 104  
    Algorithm::DoublyLinkedList, 213  
    Algorithm::DynamicArray, 246  
    Algorithm::Queue, 463  
    Algorithm::SinglyLinkedList, 507  
    Algorithm::Stack, 547  
    Algorithm::StaticArrayList, 599  
Get  
    CategoryInfo, 97  
    DSInfo, 240  
    FontHolder, 284  
    TextureHolder, 629, 630  
get\_iterator  
    Core::Deque< T >, 188  
    Core::List< T >, 366  
    Core::Queue< T >, 476  
    Core::Stack< T >, 561  
GetActionDescription  
    Animation::AnimationState< T >, 50  
GetAnimateFrame  
    Animation::AnimationController< T >, 40  
GetAnimation  
    Animation::AnimationController< T >, 41  
GetAnimationDuration  
    Animation::AnimationController< T >, 41

GetAnimationIndex  
     Animation::AnimationController< T >, 41

GetArrowType  
     GUIVisualizer::CircularLinkedList, 119  
     GUIVisualizer::SinglyLinkedList, 522

GetArrowTypeNext  
     GUIVisualizer::DoublyLinkedList, 229

GetArrowTypePrev  
     GUIVisualizer::DoublyLinkedList, 229

GetBackgroundColor  
     GUIVisualizer::Node, 410

GetCapacity  
     GUIVisualizer::DynamicArray, 260

GetCapacityFromLength  
     GUIVisualizer::DynamicArray, 260

GetChildren  
     GUI::Container, 172  
     GUIComponent::Footer< T >, 291  
     GUIComponent::OperationContainer, 426  
     GUIComponent::OperationList, 437  
     GUIComponent::OptionInputField, 449  
     GUIVisualizer::CircularLinkedList, 119  
     GUIVisualizer::DoublyLinkedList, 229  
     GUIVisualizer::DynamicArray, 260  
     GUIVisualizer::LinkedList, 348  
     GUIVisualizer::SinglyLinkedList, 522

GetCircularArrowType  
     GUIVisualizer::CircularLinkedList, 119

GetCircularEnds  
     GUIVisualizer::CircularLinkedList, 119

getColor  
     Settings, 499

GetContentPos  
     GUIComponent::Button, 77

GetContentSize  
     GUIComponent::Button, 77

GetContext  
     State::ArrayState< T >, 66  
     State::CLLState, 135  
     State::DLLState, 203  
     State::DynamicArrayState, 278  
     State::HomepageState, 301  
     State::LLState< T >, 382  
     State::QueueState, 493  
     State::SLLState, 538  
     State::StackState, 578  
     State::State, 585  
     State::StaticArrayState, 611

GetCurrentPlayingAt  
     Animation::AnimationState< T >, 50

GetDataStructure  
     Animation::AnimationState< T >, 50

GetDuration  
     Animation::AnimationState< T >, 51

GetEditMode  
     GUIComponent::InputField, 313  
     GUIComponent::IntegerInputField, 327  
     GUIComponent::StringInputField, 620

GetFontSize  
     GUIComponent::Button, 77

GetHighlightedLine  
     Animation::AnimationState< T >, 51

GetHoverStatus  
     GUI::Component, 154, 155  
     GUI::Container, 172  
     GUIComponent::Button, 77, 78  
     GUIComponent::Card, 90  
     GUIComponent::CodeHighlighter, 144  
     GUIComponent::Footer< T >, 291  
     GUIComponent::InputField, 313  
     GUIComponent::IntegerInputField, 327  
     GUIComponent::NavigationBar, 392  
     GUIComponent::OperationContainer, 426, 427  
     GUIComponent::OperationList, 437  
     GUIComponent::OptionInputField, 449, 450  
     GUIComponent::StringInputField, 620  
     GUIVisualizer::CircularLinkedList, 119, 120  
     GUIVisualizer::DoublyLinkedList, 229, 230  
     GUIVisualizer::DynamicArray, 260, 261  
     GUIVisualizer::LinkedList, 348, 349  
     GUIVisualizer::Node, 410  
     GUIVisualizer::SinglyLinkedList, 522

getInstance  
     Settings, 500

GetLabel  
     GUIComponent::InputField, 314  
     GUIComponent::IntegerInputField, 328  
     GUIComponent::StringInputField, 621

GetLength  
     GUIVisualizer::DynamicArray, 261

GetList  
     GUIVisualizer::CircularLinkedList, 120  
     GUIVisualizer::DoublyLinkedList, 230  
     GUIVisualizer::DynamicArray, 261  
     GUIVisualizer::LinkedList, 349  
     GUIVisualizer::SinglyLinkedList, 523

GetNodeDefaultPosition  
     GUIVisualizer::CircularLinkedList, 120  
     GUIVisualizer::DoublyLinkedList, 230  
     GUIVisualizer::DynamicArray, 261  
     GUIVisualizer::LinkedList, 349  
     GUIVisualizer::SinglyLinkedList, 523

GetNodeState  
     GUIVisualizer::Node, 411

GetNumAnimation  
     Animation::AnimationController< T >, 41

GetOutlineColor  
     GUIVisualizer::Node, 411

GetPosition  
     GUI::Component, 155  
     GUI::Container, 173  
     GUIComponent::Button, 78  
     GUIComponent::Card, 90  
     GUIComponent::CodeHighlighter, 145  
     GUIComponent::Footer< T >, 292  
     GUIComponent::InputField, 314

GUIComponent::IntegerField, 328  
    GUIComponent::NavigationBar, 393  
    GUIComponent::OperationContainer, 427  
    GUIComponent::OperationList, 438  
    GUIComponent::OptionInputField, 450  
    GUIComponent::StringInputField, 621  
    GUIVisualizer::CircularLinkedList, 120  
    GUIVisualizer::DoublyLinkedList, 230  
    GUIVisualizer::DynamicArray, 262  
    GUIVisualizer::LinkedList, 349  
    GUIVisualizer::Node, 411  
    GUIVisualizer::SinglyLinkedList, 523  
GetReachable  
    GUIVisualizer::Node, 411  
GetShape  
    GUIVisualizer::CircularLinkedList, 120  
    GUIVisualizer::DoublyLinkedList, 230  
    GUIVisualizer::DynamicArray, 262  
    GUIVisualizer::LinkedList, 350  
    GUIVisualizer::Node, 411  
    GUIVisualizer::SinglyLinkedList, 523  
GetSize  
    GUI::Component, 155  
    GUI::Container, 173  
    GUIComponent::Button, 78  
    GUIComponent::Card, 91  
    GUIComponent::CodeHighlighter, 145  
    GUIComponent::Footer< T >, 292  
    GUIComponent::InputField, 314  
    GUIComponent::IntegerField, 328  
    GUIComponent::NavigationBar, 393  
    GUIComponent::OperationContainer, 427  
    GUIComponent::OperationList, 438  
    GUIComponent::OptionInputField, 450  
    GUIComponent::StringInputField, 621  
    GUIVisualizer::CircularLinkedList, 121  
    GUIVisualizer::DoublyLinkedList, 231  
    GUIVisualizer::DynamicArray, 262  
    GUIVisualizer::LinkedList, 350  
    GUIVisualizer::Node, 411  
    GUIVisualizer::SinglyLinkedList, 523  
GetSpeed  
    Animation::AnimationController< T >, 42  
GetStopDuration  
    Animation::AnimationController< T >, 42  
GetTextColor  
    GUIVisualizer::Node, 412  
GetValue  
    GUIVisualizer::Node, 412  
GetVisible  
    GUI::Component, 155  
    GUI::Container, 173  
    GUIComponent::Button, 78  
    GUIComponent::Card, 91  
    GUIComponent::CodeHighlighter, 145  
    GUIComponent::Footer< T >, 292  
    GUIComponent::InputField, 314  
    GUIComponent::IntegerField, 328  
    GUIComponent::NavigationBar, 393  
    GUIComponent::OperationContainer, 428  
    GUIComponent::OperationList, 438  
    GUIComponent::OptionInputField, 450  
    GUIComponent::StringInputField, 621  
    GUIVisualizer::CircularLinkedList, 121  
    GUIVisualizer::DoublyLinkedList, 231  
    GUIVisualizer::DynamicArray, 262  
    GUIVisualizer::LinkedList, 350  
    GUIVisualizer::Node, 412  
    GUIVisualizer::SinglyLinkedList, 524  
global, 23  
    favicon, 23  
    kTitle, 23  
    SCREEN\_HEIGHT, 23  
    SCREEN\_WIDTH, 23  
GUI, 23  
GUI::Component, 151  
    ~Component, 153  
    Component, 153  
    deselect, 154  
    Draw, 154  
    GetHoverStatus, 154, 155  
    GetPosition, 155  
    GetSize, 155  
    GetVisible, 155  
    isSelectable, 156  
    isSelected, 156  
    mIsSelected, 160  
    mPosition, 160  
    mVisible, 160  
    Ptr, 153  
    select, 156  
    SetPosition, 157  
    SetVisible, 157  
    ToggleVisible, 159  
    UpdateMouseCursorWhenHover, 159  
GUI::Container, 168  
    Container, 171  
    deselect, 171  
    Draw, 171  
    DrawCurrent, 172  
    GetChildren, 172  
    GetHoverStatus, 172  
    GetPosition, 173  
    GetSize, 173  
    GetVisible, 173  
    isSelectable, 174  
    isSelected, 174  
    mChildren, 177  
    mIsSelected, 177  
    mPosition, 177  
    mVisible, 177  
    pack, 174  
    Ptr, 170  
    select, 175  
    SetPosition, 175  
    SetVisible, 176

ToggleVisible, 176  
UnpackAll, 176  
UpdateMouseCursorWhenHover, 176, 177  
**GUIComponent**, 24  
**GUIComponent::Button**, 71  
    ~Button, 75  
    action, 83  
    alignment, 83  
    AlignmentCount, 75  
    bound, 83  
    Button, 75  
    buttonColorTheme, 83  
    buttonHoverColorTheme, 83  
    Center, 75  
    content, 83  
    deselect, 75  
    DisableFitContent, 75  
    Draw, 76  
    DrawButtonText, 77  
    EnableFitContent, 77  
    FitContent, 77  
    fitContent, 84  
    fonts, 84  
    fontSize, 84  
    GetContentPos, 77  
    GetContentSize, 77  
    GetFontSize, 77  
    GetHoverStatus, 77, 78  
    GetPosition, 78  
    GetSize, 78  
    GetVisible, 78  
    IsClicked, 79  
    isHover, 84  
    isSelectable, 79  
    isSelected, 79  
    Left, 75  
    mActionOnHover, 84  
    mIsSelected, 84  
    mPosition, 84  
    mVisible, 85  
    Ptr, 74  
    Right, 75  
    select, 79  
    SetAction, 80  
    SetActionOnHover, 80  
    SetButtonColor, 80  
    SetButtonHoverColor, 80  
    SetFontSize, 80  
    SetPosition, 80, 81  
    SetSize, 81  
    SetText, 81  
    SetTextAlignment, 81  
    SetTextColor, 81  
    SetVisible, 82  
    TextAlignment, 74  
    textColorTheme, 85  
    ToggleVisible, 82  
    UpdateMouseCursorWhenHover, 82  
**GUIComponent::Card**, 85  
    ~Card, 89  
    Card, 88  
    deselect, 89  
    Draw, 89  
    DrawImage, 89  
    DrawTitle, 89  
    fonts, 94  
    GetHoverStatus, 90  
    GetPosition, 90  
    GetSize, 91  
    GetVisible, 91  
    hoverBounds, 94  
    isHover, 95  
    isSelectable, 91  
    isSelected, 91  
    mIsSelected, 95  
    mPosition, 95  
    mVisible, 95  
    Ptr, 88  
    select, 92  
    SetLink, 92  
    SetPosition, 92, 93  
    SetStateID, 93  
    SetText, 93  
    SetVisible, 93  
    stateID, 95  
    thumbnail, 95  
    title, 96  
    ToggleVisible, 93  
    toLink, 96  
    UpdateMouseCursorWhenHover, 94  
**GUIComponent::CodeHighlighter**, 140  
    ~CodeHighlighter, 143  
    AddActionDescription, 143  
    AddCode, 143  
    CodeHighlighter, 142  
    deselect, 143  
    Draw, 143  
    DrawActionDescription, 144  
    DrawCodeHighlighter, 144  
    fonts, 149  
    GetHoverStatus, 144  
    GetPosition, 145  
    GetSize, 145  
    GetVisible, 145  
    Highlight, 145  
    InitButtons, 146  
    isSelectable, 146  
    isSelected, 146  
    mActionDescription, 149  
    mButtonShowAction, 149  
    mButtonShowCode, 149  
    mCode, 150  
    mHighlightedLine, 150  
    mIsSelected, 150  
    mPosition, 150  
    mShowActionDescription, 150

mShowCode, 150  
mVisible, 150  
Ptr, 142  
select, 146  
SetPosition, 147  
SetShowAction, 147  
SetShowCode, 147  
SetVisible, 148  
ToggleShowAction, 148  
ToggleShowCode, 148  
ToggleVisible, 148  
UpdateMouseCursorWhenHover, 148, 149

GUIComponent::Footer< T >, 286  
    ~Footer, 289  
    deselect, 290  
    Draw, 290  
    DrawCurrent, 290  
    Footer, 289  
    GetChildren, 291  
    GetHoverStatus, 291  
    GetPosition, 292  
    GetSize, 292  
    GetVisible, 292  
    isSelectable, 292  
    isSelected, 293  
    mChildren, 296  
    mIsSelected, 296  
    mPosition, 296  
    mVisible, 296  
    pack, 293  
    Ptr, 289  
    select, 294  
    SetPosition, 294  
    SetVisible, 295  
    ToggleVisible, 295  
    UnpackAll, 295  
    UpdateMouseCursorWhenHover, 295, 296

GUIComponent::InputField, 309  
    ~InputField, 311  
    AllFieldDisableEdit, 312  
    deselect, 312  
    Draw, 312  
    DrawField, 312  
    editMode, 318  
    extractedValue, 318  
    ExtractValue, 313  
    fields, 318  
    fonts, 319  
    GetEditMode, 313  
    GetHoverStatus, 313  
    GetLabel, 314  
    GetPosition, 314  
    GetSize, 314  
    GetVisible, 314  
    InputField, 311  
    inputFontSize, 319  
    IsClicked, 314  
    isSelectable, 315  
    isSelected, 315  
    label, 319  
    labelFontSize, 319  
    mFieldIndex, 319  
    mIsSelected, 319  
    mPosition, 319  
    mVisible, 320  
    Ptr, 311  
    Randomize, 315  
    select, 315  
    SetEditMode, 316  
    SetInputFontSize, 316  
    SetLabel, 316  
    SetLabelSize, 316  
    SetPosition, 316, 317  
    SetVisible, 317  
    ToggleVisible, 317  
    UpdateMouseCursorWhenHover, 317, 318

GUIComponent::IntegerField, 322  
    ~IntegerField, 325  
    AllFieldDisableEdit, 326  
    deselect, 326  
    Draw, 326  
    DrawField, 326  
    editMode, 332  
    extractedValue, 332  
    ExtractValue, 327  
    fields, 332  
    fonts, 333  
    GetEditMode, 327  
    GetHoverStatus, 327  
    GetLabel, 328  
    GetPosition, 328  
    GetSize, 328  
    GetVisible, 328  
    input, 333  
    inputFontSize, 333  
    IntegerField, 325  
    IsClicked, 328  
    isSelectable, 329  
    isSelected, 329  
    label, 333  
    labelFontSize, 333  
    mFieldIndex, 333  
    mIsSelected, 333  
    mMaxValue, 334  
    mMinValue, 334  
    mPosition, 334  
    mVisible, 334  
    Ptr, 325  
    Randomize, 329  
    select, 329  
    SetConstraint, 330  
    SetEditMode, 330  
    SetInputFontSize, 330  
    SetLabel, 330  
    SetLabelSize, 330  
    SetPosition, 330, 331

SetVisible, 331  
 ToggleVisible, 331  
 UpdateMouseCursorWhenHover, 331, 332  
**GUIComponent::NavigationBar**, 387  
 ~NavigationBar, 391  
 activeTitle, 397  
 ClearTitle, 391  
 currentCategory, 397  
 deselect, 391  
 Draw, 391  
 DrawLogo, 392  
 DrawSwitchTheme, 392  
 DrawTitles, 392  
 fonts, 398  
 GetHoverStatus, 392  
 GetPosition, 393  
 GetSize, 393  
 GetVisible, 393  
 hasTitle, 398  
 homepageID, 398  
 hoverBounds, 398  
 InsertTitle, 393  
 isHover, 398  
 isSelectable, 394  
 isSelected, 394  
 mIsSelected, 398  
 mPosition, 398  
 mTitles, 399  
 mVisible, 399  
 NavigationBar, 391  
 Ptr, 390  
 select, 394  
 SetActiveTitle, 395  
 SetCategory, 395  
 SetDirectLink, 395  
 SetHomepageID, 395  
 SetPosition, 395, 396  
 SetVisibleTitle, 396  
 setVisible, 396  
 ToggleVisible, 396  
 toLink, 399  
 UpdateMouseCursorWhenHover, 397  
**GUIComponent::NavigationBar::TitleInfo**, 633  
 abbrTitle, 633  
 stateID, 633  
 titleName, 633  
**GUIComponent::OperationContainer**, 422  
 ~OperationContainer, 425  
 deselect, 425  
 Draw, 426  
 DrawCurrent, 426  
 GetChildren, 426  
 GetHoverStatus, 426, 427  
 GetPosition, 427  
 GetSize, 427  
 GetVisible, 428  
 isSelectable, 428  
 isSelected, 428  
 mChildren, 432  
 mIsSelected, 432  
 mPosition, 432  
 mVisible, 432  
 OperationContainer, 425  
 pack, 429  
 Ptr, 425  
 select, 429  
 SetPosition, 429, 430  
 SetVisible, 430  
 ToggleVisible, 430  
 UnpackAll, 430  
 UpdateMouseCursorWhenHover, 431  
 UpdatePosition, 431  
**GUIComponent::OperationList**, 433  
 ~OperationList, 436  
 AddOperation, 436  
 buttons, 443  
 deselect, 436  
 Draw, 436  
 DrawCurrent, 437  
 GetChildren, 437  
 GetHoverStatus, 437  
 GetPosition, 438  
 GetSize, 438  
 GetVisible, 438  
 HideAllOptions, 439  
 InitActionBar, 439  
 isHide, 443  
 isSelectable, 439  
 isSelected, 439  
 mChildren, 443  
 mIsSelected, 443  
 mPosition, 443  
 mVisible, 444  
 OperationList, 436  
 optionContainers, 444  
 pack, 440  
 Ptr, 435  
 select, 440  
 SetPosition, 440, 441  
 SetVisible, 441  
 ShowOptions, 441  
 toggleButton, 444  
 ToggleOperations, 441  
 ToggleVisible, 442  
 UnpackAll, 442  
 UpdateMouseCursorWhenHover, 442  
**GUIComponent::OptionInputField**, 444  
 ~OptionInputField, 448  
 AddInputField, 448  
 AddSubmitField, 448  
 deselect, 448  
 Draw, 448  
 DrawCurrent, 449  
 ExtractInput, 449  
 fonts, 454  
 GetChildren, 449

GetHoverStatus, 449, 450  
GetPosition, 450  
GetSize, 450  
GetVisible, 450  
HasInputField, 451  
hasInputField, 455  
isSelectable, 451  
isSelected, 451  
mChildren, 455  
mInput, 455  
mInputField, 455  
mIsSelected, 455  
mPosition, 455  
mVisible, 455  
OptionInputField, 447  
pack, 451  
Ptr, 447  
select, 452  
SetNoFieldOption, 452  
SetOption, 452  
SetPosition, 452, 453  
SetVisible, 453  
ToggleInputFields, 453  
ToggleVisible, 453  
UnpackAll, 453  
UpdateMouseCursorWhenHover, 454  
GUIComponent::StringInputField, 615  
  ~StringInputField, 618  
  AllFieldDisableEdit, 619  
  content, 625  
  deselect, 619  
  Draw, 619  
  DrawField, 619  
  editMode, 625  
  extractedValue, 625  
  ExtractValue, 620  
  fields, 626  
  fonts, 626  
  GetEditMode, 620  
  GetHoverStatus, 620  
  GetLabel, 621  
  GetPosition, 621  
  GetSize, 621  
  GetVisible, 621  
  inputFieldSize, 626  
  IsClicked, 621  
  isSelectable, 622  
  isSelected, 622  
  label, 626  
  labelFontSize, 626  
  mFieldIndex, 626  
  mIsSelected, 626  
  mMaxLength, 627  
  mPosition, 627  
  mVisible, 627  
  Ptr, 618  
  Randomize, 622  
  select, 622  
    SetEditMode, 623  
    SetInputFieldSize, 623  
    SetLabel, 623  
    SetLabelSize, 623  
    SetPosition, 623, 624  
    SetVisible, 624  
    StringInputField, 618  
    ToggleVisible, 624  
    UpdateMouseCursorWhenHover, 624, 625  
  GUIVisualizer, 24  
  GUIVisualizer::CircularLinkedList, 111  
    ~CircularLinkedList, 116  
    Active, 116  
    arrowState, 127  
    ArrowType, 115  
    ArrowTypeCount, 116  
    circularArrowState, 127  
    CircularLinkedList, 116  
    Default, 116  
    DeleteNode, 117  
    deselect, 117  
    Draw, 117, 118  
    DrawArrow, 118  
    DrawCurrent, 118  
    fonts, 127  
    GenerateNode, 118  
    GetArrowType, 119  
    GetChildren, 119  
    GetCircularArrowType, 119  
    GetCircularEnds, 119  
    GetHoverStatus, 119, 120  
    GetList, 120  
    GetNodeDefaultPosition, 120  
    GetPosition, 120  
    GetShape, 120  
    GetSize, 121  
    GetVisible, 121  
    Hidden, 116  
    Horizontal, 116  
    Import, 121  
    InsertNode, 122  
    isSelectable, 122  
    isSelected, 122  
    list, 127  
    mChildren, 128  
    mCircularEnds, 128  
    mDisplayHeadAndTail, 128  
    mIsSelected, 128  
    mNodeDistance, 128  
    mOrientation, 128  
    mPosition, 129  
    mShape, 129  
    mVisible, 129  
    Orientation, 116  
    OrientationCount, 116  
    pack, 123  
    Ptr, 115  
    Relayout, 123

ResetArrow, 123  
 select, 123  
 SetArrowType, 124  
 SetCircularArrowType, 124  
 SetCircularEnds, 124  
 SetOrientation, 124  
 SetPosition, 124, 125  
 SetShape, 125  
 SetShowHeadAndTail, 125  
 SetVisible, 125  
 size, 126  
 Skip, 116  
 ToggleVisible, 126  
 UnpackAll, 126  
 UpdateMouseCursorWhenHover, 126, 127  
 Vertical, 116  
**GUIVisualizer::DoublyLinkedList**, 221  
 ~DoublyLinkedList, 226  
 Active, 226  
 arrowStateNext, 237  
 arrowStatePrev, 237  
 ArrowType, 225  
 ArrowTypeCount, 226  
 Default, 226  
 DeleteNode, 227  
 deselect, 227  
 DoublyLinkedList, 226  
 Draw, 227, 228  
 DrawArrow, 228  
 DrawCurrent, 228  
 fonts, 237  
 GenerateNode, 228  
 GetArrowTypeNext, 229  
 GetArrowTypePrev, 229  
 GetChildren, 229  
 GetHoverStatus, 229, 230  
 GetList, 230  
 GetNodeDefaultPosition, 230  
 GetPosition, 230  
 GetShape, 230  
 GetSize, 231  
 GetVisible, 231  
 Hidden, 226  
 Horizontal, 226  
 Import, 231  
 InsertNode, 232  
 isSelectable, 232  
 isSelected, 232  
 list, 237  
 mChildren, 238  
 mDisplayHeadAndTail, 238  
 mIsSelected, 238  
 mNodeDistance, 238  
 mOrientation, 238  
 mPosition, 238  
 mShape, 239  
 mVisible, 239  
 Orientation, 226  
 OrientationCount, 226  
 pack, 233  
 Ptr, 225  
 Relayout, 233  
 ResetArrow, 233  
 select, 233  
 SetArrowTypeNext, 234  
 SetArrowTypePrev, 234  
 SetOrientation, 234  
 SetPosition, 234, 235  
 SetShape, 235  
 SetShowHeadAndTail, 235  
 SetVisible, 235  
 size, 236  
 Skip, 226  
 ToggleVisible, 236  
 UnpackAll, 236  
 UpdateMouseCursorWhenHover, 236, 237  
 Vertical, 226  
**GUIVisualizer::DynamicArray**, 253  
 ~DynamicArray, 258  
 capacity, 270  
 Clear, 258  
 DeleteNode, 258  
 deselect, 258  
 Draw, 259  
 DrawCurrent, 259  
 DynamicArray, 257  
 fonts, 270  
 GenerateNode, 259  
 GetCapacity, 260  
 GetCapacityFromLength, 260  
 GetChildren, 260  
 GetHoverStatus, 260, 261  
 GetLength, 261  
 GetList, 261  
 GetNodeDefaultPosition, 261  
 GetPosition, 262  
 GetShape, 262  
 GetSize, 262  
 GetVisible, 262  
 Import, 263  
 InsertNode, 263  
 isSelectable, 263  
 isSelected, 264  
 length, 270  
 list, 270  
 mChildren, 270  
 mIsSelected, 270  
 mNodeDistance, 270  
 mPosition, 271  
 mShape, 271  
 mVisible, 271  
 operator[], 264, 265  
 pack, 265  
 Ptr, 257  
 Relayout, 265  
 Reserve, 265

Resize, 267  
select, 267  
SetPosition, 267  
SetShape, 268  
SetVisible, 268  
ToggleVisible, 268  
UnpackAll, 269  
UpdateMouseCursorWhenHover, 269  
GUIVisualizer::LinkedList, 341  
~LinkedList, 346  
Active, 345  
ArrowType, 345  
ArrowTypeCount, 345  
Default, 345  
DeleteNode, 347  
deselect, 347  
Draw, 347, 348  
DrawCurrent, 348  
fonts, 355  
GenerateNode, 348  
GetChildren, 348  
GetHoverStatus, 348, 349  
GetList, 349  
GetNodeDefaultPosition, 349  
GetPosition, 349  
GetShape, 350  
GetSize, 350  
GetVisible, 350  
Hidden, 345  
Horizontal, 346  
Import, 350  
InsertNode, 351  
isSelectable, 351  
isSelected, 351  
LinkedList, 346  
list, 356  
mChildren, 356  
mDisplayHeadAndTail, 356  
mIsSelected, 356  
mNodeDistance, 356  
mOrientation, 356  
mPosition, 357  
mShape, 357  
mVisible, 357  
Orientation, 346  
OrientationCount, 346  
pack, 352  
Ptr, 345  
Relayout, 352  
select, 352  
SetOrientation, 352  
SetPosition, 353  
SetShape, 353  
SetShowHeadAndTail, 354  
SetVisible, 354  
size, 354  
Skip, 345  
ToggleVisible, 354  
UnpackAll, 354  
UpdateMouseCursorWhenHover, 355  
Vertical, 346  
GUIVisualizer::Node, 402  
~Node, 408  
Active, 407  
ActiveBlue, 407  
ActiveGreen, 407  
ActiveRed, 407  
AddColor, 408  
animateNode, 417  
AnimationOnNode, 408  
Circle, 407  
ClearLabel, 408  
Default, 407  
deselect, 408  
Draw, 409  
DrawLabel, 409  
DrawNode, 409  
fonts, 417  
GetBackgroundColor, 410  
GetHoverStatus, 410  
GetNodeState, 411  
GetOutlineColor, 411  
GetPosition, 411  
GetReachable, 411  
GetShape, 411  
GetSize, 411  
GetTextColor, 412  
GetValue, 412  
GetVisible, 412  
Hide, 407  
IsActive, 412  
isSelectable, 413  
isSelected, 413  
Iterated, 407  
labelFontSize, 417  
mActive, 417  
mActiveColor, 418  
mBackgroundColor, 418  
mBorderColor, 418  
mDefaultColor, 418  
mIsSelected, 418  
mLabel, 418  
mNodeState, 418  
mOutlineColor, 419  
mPosition, 419  
mRadius, 419  
mReachable, 419  
mShape, 419  
mTextColor, 419  
mValue, 419  
mVisible, 420  
Node, 407  
Ptr, 406  
select, 413  
SetActive, 413  
SetLabel, 414

SetLabelFontSize, 414  
 SetNodeState, 414  
 SetPosition, 414, 415  
 SetRadius, 415  
 SetReachable, 415  
 SetShape, 415  
 SetValue, 415  
 SetValueFontSize, 416  
 SetVisible, 416  
 Shape, 406  
 ShapeCount, 407  
 Square, 407  
 State, 407  
 StateCount, 407  
 ToggleVisible, 416  
 UpdateMouseCursorWhenHover, 416, 417  
 valueFontSize, 420  
**GUIVisualizer::SinglyLinkedList**, 514  
 ~SinglyLinkedList, 519  
 Active, 519  
 arrowState, 529  
 ArrowType, 518  
 ArrowTypeCount, 519  
 Default, 519  
 DeleteNode, 520  
 deselect, 520  
 Draw, 520, 521  
 DrawArrow, 521  
 DrawCurrent, 521  
 fonts, 530  
 GenerateNode, 521  
 GetArrowType, 522  
 GetChildren, 522  
 GetHoverStatus, 522  
 GetList, 523  
 GetNodeDefaultPosition, 523  
 GetPosition, 523  
 GetShape, 523  
 GetSize, 523  
 GetVisible, 524  
 Hidden, 519  
 Horizontal, 519  
 Import, 524  
 InsertNode, 524  
 isSelectable, 525  
 isSelected, 525  
 list, 530  
 mChildren, 530  
 mDisplayHeadAndTail, 530  
 mIsSelected, 530  
 mNodeDistance, 530  
 mOrientation, 530  
 mPosition, 531  
 mShape, 531  
 mVisible, 531  
 Orientation, 519  
 OrientationCount, 519  
 pack, 525  
 Ptr, 518  
 Relayout, 526  
 ResetArrow, 526  
 select, 526  
 SetArrowType, 526  
 SetOrientation, 526  
 SetPosition, 527  
 SetShape, 527  
 SetShowHeadAndTail, 527  
 SetVisible, 528  
 SinglyLinkedList, 519  
 size, 528  
 Skip, 519  
 ToggleVisible, 528  
 UnpackAll, 528  
 UpdateMouseCursorWhenHover, 529  
 Vertical, 519  
**hasInitializeCard**  
 State::HomepageState, 303  
**HasInputField**  
 GUIComponent::OptionInputField, 451  
**hasInputField**  
 GUIComponent::OptionInputField, 455  
**hasTitle**  
 GUIComponent::NavigationBar, 398  
**Hidden**  
 GUIVisualizer::CircularLinkedList, 116  
 GUIVisualizer::DoublyLinkedList, 226  
 GUIVisualizer::LinkedList, 345  
 GUIVisualizer::SinglyLinkedList, 519  
**Hide**  
 GUIVisualizer::Node, 407  
**HideAllOptions**  
 GUIComponent::OperationList, 439  
**Highlight**  
 GUIComponent::CodeHighlighter, 145  
**HighlightArrowBoth**  
 Algorithm::DoublyLinkedList, 214  
**HighlightArrowFromCur**  
 Algorithm::CircularLinkedList, 105  
 Algorithm::Queue, 463  
 Algorithm::SinglyLinkedList, 508  
 Algorithm::Stack, 547  
**HighlightArrowNext**  
 Algorithm::DoublyLinkedList, 214  
**HighlightArrowPrev**  
 Algorithm::DoublyLinkedList, 214  
**HighlightCircularArrow**  
 Algorithm::CircularLinkedList, 105  
**Homepage**  
 States, 26  
**homepageID**  
 GUIComponent::NavigationBar, 398  
**HomepageState**  
 State::HomepageState, 300  
**Horizontal**  
 GUIVisualizer::CircularLinkedList, 116  
 GUIVisualizer::DoublyLinkedList, 226

GUIVisualizer::LinkedList, 346  
    GUIVisualizer::SinglyLinkedList, 519

hoverBounds  
    GUIComponent::Card, 94  
    GUIComponent::NavigationBar, 398

ID  
    Category, 18  
    ColorTheme, 19  
    DataStructures, 22  
    DSInfo::Info, 307  
    Fonts, 22  
    States, 25  
    Textures, 26

images  
    Application, 59

Import  
    GUIVisualizer::CircularLinkedList, 121  
    GUIVisualizer::DoublyLinkedList, 231  
    GUIVisualizer::DynamicArray, 263  
    GUIVisualizer::LinkedList, 350  
    GUIVisualizer::SinglyLinkedList, 524

Info  
    CategoryInfo::Info, 304  
    DSInfo::Info, 306

Init  
    Application, 56

InitAction  
    Algorithm::Algorithm< GUIAlgorithm, Animation-  
        State >, 33  
    Algorithm::CircularLinkedList, 105  
    Algorithm::DoublyLinkedList, 215  
    Algorithm::DynamicArray, 247  
    Algorithm::Queue, 463  
    Algorithm::SinglyLinkedList, 508  
    Algorithm::Stack, 548  
    Algorithm::StaticArray, 599

InitActionBar  
    GUIComponent::OperationList, 439

InitButtons  
    GUIComponent::CodeHighlighter, 146

InitCards  
    State::HomepageState, 301

InitNavigationBar  
    State::ArrayList< T >, 67  
    State::CLLState, 136  
    State::DLLState, 204  
    State::DynamicArrayList, 278  
    State::HomepageState, 301  
    State::LLState< T >, 382  
    State::QueueState, 493  
    State::SLLState, 538  
    State::StackState, 578  
    State::State, 585  
    State::StaticArrayList, 611

input  
    GUIComponent::IntegerField, 333

InputField  
    GUIComponent::InputField, 311

    InputField\_Inactive  
        ColorTheme, 20

inputFontSize  
    GUIComponent::InputField, 319  
    GUIComponent::IntegerField, 333  
    GUIComponent::StringInputField, 626

Insert  
    Algorithm::StaticArray, 599

insert\_previous  
    Core::Deque< T >, 188  
    Core::List< T >, 367  
    Core::Queue< T >, 476  
    Core::Stack< T >, 561

InsertAfterTail  
    Algorithm::CircularLinkedList, 106  
    Algorithm::DoublyLinkedList, 215  
    Algorithm::SinglyLinkedList, 508

InsertHead  
    Algorithm::CircularLinkedList, 106  
    Algorithm::DoublyLinkedList, 215  
    Algorithm::SinglyLinkedList, 509

InsertMiddle  
    Algorithm::CircularLinkedList, 106  
    Algorithm::DoublyLinkedList, 216  
    Algorithm::SinglyLinkedList, 509

InsertNode  
    GUIVisualizer::CircularLinkedList, 122  
    GUIVisualizer::DoublyLinkedList, 232  
    GUIVisualizer::DynamicArray, 263  
    GUIVisualizer::LinkedList, 351  
    GUIVisualizer::SinglyLinkedList, 524

InsertResource  
    FontHolder, 285  
    TextureHolder, 630

InsertTitle  
    GUIComponent::NavigationBar, 393

IntegerField  
    GUIComponent::IntegerField, 325

InteractionAllow  
    Animation::AnimationController< T >, 42

InteractionLock  
    Animation::AnimationController< T >, 42

InverseVector  
    AnimationFactory, 18

IsActive  
    GUIVisualizer::Node, 412

IsClicked  
    GUIComponent::Button, 79  
    GUIComponent::InputField, 314  
    GUIComponent::IntegerField, 328  
    GUIComponent::StringInputField, 621

IsEmpty  
    State::StateStack, 591

isHide  
    GUIComponent::OperationList, 443

isHover  
    GUIComponent::Button, 84  
    GUIComponent::Card, 95

**GUIComponent::NavigationBar**, 398  
**IsInteractionAllow**  
  **Animation::AnimationController< T >**, 43  
**IsPlaying**  
  **Animation::AnimationController< T >**, 43  
**isSelectable**  
  **GUI::Component**, 156  
  **GUI::Container**, 174  
  **GUIComponent::Button**, 79  
  **GUIComponent::Card**, 91  
  **GUIComponent::CodeHighlighter**, 146  
  **GUIComponent::Footer< T >**, 292  
  **GUIComponent::InputField**, 315  
  **GUIComponent::IntegerInputField**, 329  
  **GUIComponent::NavigationBar**, 394  
  **GUIComponent::OperationContainer**, 428  
  **GUIComponent::OperationList**, 439  
  **GUIComponent::OptionInputField**, 451  
  **GUIComponent::StringInputField**, 622  
**GUIVisualizer::CircularLinkedList**, 122  
**GUIVisualizer::DoublyLinkedList**, 232  
**GUIVisualizer::DynamicArray**, 263  
**GUIVisualizer::LinkedList**, 351  
**GUIVisualizer::Node**, 413  
**GUIVisualizer::SinglyLinkedList**, 525  
**isSelected**  
  **GUI::Component**, 156  
  **GUI::Container**, 174  
  **GUIComponent::Button**, 79  
  **GUIComponent::Card**, 91  
  **GUIComponent::CodeHighlighter**, 146  
  **GUIComponent::Footer< T >**, 293  
  **GUIComponent::InputField**, 315  
  **GUIComponent::IntegerInputField**, 329  
  **GUIComponent::NavigationBar**, 394  
  **GUIComponent::OperationContainer**, 428  
  **GUIComponent::OperationList**, 439  
  **GUIComponent::OptionInputField**, 451  
  **GUIComponent::StringInputField**, 622  
**GUIVisualizer::CircularLinkedList**, 122  
**GUIVisualizer::DoublyLinkedList**, 232  
**GUIVisualizer::DynamicArray**, 264  
**GUIVisualizer::LinkedList**, 351  
**GUIVisualizer::Node**, 413  
**GUIVisualizer::SinglyLinkedList**, 525  
**Iterated**  
  **GUIVisualizer::Node**, 407  
**iterator**  
  **Core::List< T >::iterator**, 336, 337  
**iterator\_category**  
  **Core::List< T >::const\_iterator**, 162  
  **Core::List< T >::iterator**, 336  
  
**kTitle**  
  **global**, 23  
  
**label**  
  **GUIComponent::InputField**, 319  
  **GUIComponent::IntegerInputField**, 333  
  
**GUIComponent::StringInputField**, 626  
**State::ArrayState< T >::IntegerInput**, 320  
**State::LLState< T >::IntegerInput**, 321  
**labelFontSize**  
  **GUIComponent::InputField**, 319  
  **GUIComponent::IntegerInputField**, 333  
  **GUIComponent::StringInputField**, 626  
  **GUIVisualizer::Node**, 417  
**Left**  
  **GUIComponent::Button**, 75  
**length**  
  **GUIVisualizer::DynamicArray**, 270  
**LinkedList**  
  **Category**, 19  
  **GUIVisualizer::LinkedList**, 346  
**List**  
  **Core::Deque< T >**, 183, 188, 189  
  **Core::List< T >**, 361, 362  
  **Core::List< T >::const\_iterator**, 168  
  **Core::List< T >::iterator**, 340  
  **Core::Queue< T >**, 472, 477, 478  
  **Core::Stack< T >**, 556, 562  
**list**  
  **GUIVisualizer::CircularLinkedList**, 127  
  **GUIVisualizer::DoublyLinkedList**, 237  
  **GUIVisualizer::DynamicArray**, 270  
  **GUIVisualizer::LinkedList**, 356  
  **GUIVisualizer::SinglyLinkedList**, 530  
**LLState**  
  **State::LLState< T >**, 377  
**Load**  
  **FontHolder**, 285  
  **Settings**, 500  
  **TextureHolder**, 631  
**LoadDarkColors**  
  **Settings**, 500  
**LoadDefaultColors**  
  **Settings**, 500  
**LoadFromImage**  
  **TextureHolder**, 632  
**LoadResources**  
  **Application**, 57  
**Logo1FirstPart**  
  **ColorTheme**, 20  
**Logo1SecondPart**  
  **ColorTheme**, 20  
**Logo2FirstPart**  
  **ColorTheme**, 20  
**Logo2SecondPart**  
  **ColorTheme**, 20  
  
**mActionDescription**  
  **GUIComponent::CodeHighlighter**, 149  
**mActionOnHover**  
  **GUIComponent::Button**, 84  
**mActive**  
  **GUIVisualizer::Node**, 417  
**mActiveColor**  
  **GUIVisualizer::Node**, 418

main  
  Main.cpp, 729

Main.cpp  
  main, 729

mAnimation  
  Animation::AnimationState< T >, 54

maxN  
  Algorithm::CircularLinkedList, 110  
  Algorithm::DoublyLinkedList, 220  
  Algorithm::DynamicArray, 252  
  Algorithm::Queue, 467  
  Algorithm::SinglyLinkedList, 513  
  Algorithm::Stack, 552  
  Algorithm::StaticArray, 604

maxValue  
  State::ArrayType< T >::IntegerInput, 320  
  State::LLState< T >::IntegerInput, 321

mBackgroundColor  
  GUVISUALIZER::Node, 418

mBegin  
  Core::Deque< T >, 196  
  Core::List< T >, 374  
  Core::Queue< T >, 486  
  Core::Stack< T >, 571

mBorderColor  
  GUVISUALIZER::Node, 418

mButtonShowAction  
  GUICOMPONENT::CodeHighlighter, 149

mButtonShowCode  
  GUICOMPONENT::CodeHighlighter, 149

mCards  
  State::HomepageState, 303

mChildren  
  GUI::Container, 177  
  GUICOMPONENT::Footer< T >, 296  
  GUICOMPONENT::OperationContainer, 432  
  GUICOMPONENT::OperationList, 443  
  GUICOMPONENT::OptionInputField, 455  
  GUVISUALIZER::CircularLinkedList, 128  
  GUVISUALIZER::DoublyLinkedList, 238  
  GUVISUALIZER::DynamicArray, 270  
  GUVISUALIZER::LinkedList, 356  
  GUVISUALIZER::SinglyLinkedList, 530

mCircularEnds  
  GUVISUALIZER::CircularLinkedList, 128

mCode  
  GUICOMPONENT::CodeHighlighter, 150

mColors  
  Settings, 501

mContext  
  State::ArrayType< T >, 70  
  State::CLLState, 138  
  State::DLLState, 206  
  State::DynamicArrayState, 281  
  State::HomepageState, 303  
  State::LLState< T >, 386  
  State::QueueState, 496  
  State::SLLState, 541

State::StackState, 581  
State::State, 587  
State::StateStack, 592  
State::StaticArrayState, 614

mCurrentAction  
  State::ArrayType< T >, 70  
  State::CLLState, 139  
  State::DLLState, 206  
  State::DynamicArrayState, 281  
  State::LLState< T >, 386  
  State::QueueState, 496  
  State::SLLState, 541  
  State::StackState, 581  
  State::StaticArrayState, 614

mCurrentAnimationIndex  
  Animation::AnimationController< T >, 46

mCurrentColorTheme  
  Settings, 501

mCurrentError  
  State::ArrayType< T >, 70  
  State::CLLState, 139  
  State::DLLState, 207  
  State::DynamicArrayState, 281  
  State::LLState< T >, 386  
  State::QueueState, 496  
  State::SLLState, 541  
  State::StackState, 581  
  State::StaticArrayState, 614

mCurrentPlayingAt  
  Animation::AnimationState< T >, 54

mCurrStopDuration  
  Animation::AnimationController< T >, 46

mDataStructureBefore  
  Animation::AnimationState< T >, 54

mDefaultColor  
  GUVISUALIZER::Node, 418

mDisplayHeadAndTail  
  GUVISUALIZER::CircularLinkedList, 128  
  GUVISUALIZER::DoublyLinkedList, 238  
  GUVISUALIZER::LinkedList, 356  
  GUVISUALIZER::SinglyLinkedList, 530

mDLL  
  State::DLLState, 207

mDS  
  CategoryInfo::Info, 305

mDuration  
  Animation::AnimationState< T >, 54

mDynamicArray  
  State::DynamicArrayState, 281

mEnd  
  Core::Deque< T >, 197  
  Core::List< T >, 374  
  Core::Queue< T >, 486  
  Core::Stack< T >, 571

mFactories  
  CategoryInfo, 98  
  DSInfo, 241  
  State::StateStack, 592

mFieldIndex  
    GUIComponent::InputField, 319  
    GUIComponent::IntegerField, 333  
    GUIComponent::StringInputField, 626

mFontMap  
    FontHolder, 286

mHighlightedLine  
    Animation::AnimationState< T >, 54  
    GUIComponent::CodeHighlighter, 150

mInput  
    GUIComponent::OptionInputField, 455

mInputField  
    GUIComponent::OptionInputField, 455

mInteractionLock  
    Animation::AnimationController< T >, 46

minValue  
    State::ArrayState< T >::IntegerInput, 321  
    State::LLState< T >::IntegerInput, 322

mIsSelected  
    GUI::Component, 160  
    GUI::Container, 177  
    GUIComponent::Button, 84  
    GUIComponent::Card, 95  
    GUIComponent::CodeHighlighter, 150  
    GUIComponent::Footer< T >, 296  
    GUIComponent::InputField, 319  
    GUIComponent::IntegerField, 333  
    GUIComponent::NavigationBar, 398  
    GUIComponent::OperationContainer, 432  
    GUIComponent::OperationList, 443  
    GUIComponent::OptionInputField, 455  
    GUIComponent::StringInputField, 626

GUIVisualizer::CircularLinkedList, 128  
GUIVisualizer::DoublyLinkedList, 238  
GUIVisualizer::DynamicArray, 270  
GUIVisualizer::LinkedList, 356  
GUIVisualizer::Node, 418  
GUIVisualizer::SinglyLinkedList, 530

mLabel  
    GUIVisualizer::Node, 418

mMaxLength  
    GUIComponent::StringInputField, 627

m.MaxValue  
    GUIComponent::IntegerField, 334

m.MinValue  
    GUIComponent::IntegerField, 334

mNext  
    Core::Node< T >, 401

mNodeDistance  
    GUIVisualizer::CircularLinkedList, 128  
    GUIVisualizer::DoublyLinkedList, 238  
    GUIVisualizer::DynamicArray, 270  
    GUIVisualizer::LinkedList, 356  
    GUIVisualizer::SinglyLinkedList, 530

mNodeState  
    GUIVisualizer::Node, 418

mOrientation  
    GUIVisualizer::CircularLinkedList, 128

mOutlineColor  
    GUIVisualizer::Node, 419

move\_previous  
    Core::Deque< T >, 189  
    Core::List< T >, 367  
    Core::Queue< T >, 478  
    Core::Stack< T >, 563

MoveNode  
    AnimationFactory, 18

mPendingList  
    State::StateStack, 593

mPlaying  
    Animation::AnimationController< T >, 46

mPosition  
    GUI::Component, 160  
    GUI::Container, 177  
    GUIComponent::Button, 84  
    GUIComponent::Card, 95  
    GUIComponent::CodeHighlighter, 150  
    GUIComponent::Footer< T >, 296  
    GUIComponent::InputField, 319  
    GUIComponent::IntegerField, 334  
    GUIComponent::NavigationBar, 398  
    GUIComponent::OperationContainer, 432  
    GUIComponent::OperationList, 443  
    GUIComponent::OptionInputField, 455  
    GUIComponent::StringInputField, 627

GUIVisualizer::CircularLinkedList, 129  
GUIVisualizer::DoublyLinkedList, 238  
GUIVisualizer::DynamicArray, 271  
GUIVisualizer::LinkedList, 357  
GUIVisualizer::Node, 419  
GUIVisualizer::SinglyLinkedList, 531

mPrev  
    Core::Node< T >, 401

mRadius  
    GUIVisualizer::Node, 419

mReachable  
    GUIVisualizer::Node, 419

mShape  
    GUIVisualizer::CircularLinkedList, 129  
    GUIVisualizer::DoublyLinkedList, 239  
    GUIVisualizer::DynamicArray, 271  
    GUIVisualizer::LinkedList, 357  
    GUIVisualizer::Node, 419  
    GUIVisualizer::SinglyLinkedList, 531

mShowActionDescription  
    GUIComponent::CodeHighlighter, 150

mShowCode  
    GUIComponent::CodeHighlighter, 150

mSize  
    Core::Deque< T >, 197  
    Core::List< T >, 374  
    Core::Queue< T >, 486  
    Core::Stack< T >, 571

mSpeed  
    Animation::AnimationController< T >, 47

mStack  
    Application, 59  
    State::ArrayState< T >, 70  
    State::CLLState, 139  
    State::DLLState, 207  
    State::DynamicArrayState, 281  
    State::HomepageState, 303  
    State::LLState< T >, 386  
    State::QueueState, 496  
    State::SLLState, 541  
    State::StackState, 581  
    State::State, 587  
    State::StateStack, 593  
    State::StaticArrayState, 614

mStackAlgorithm  
    State::StackState, 581

mStackOrientation  
    Algorithm::Stack, 552

mStaticArray  
    State::StaticArrayState, 614

mTextColor  
    GUIVisualizer::Node, 419

mTextureMap  
    TextureHolder, 633

mTitles  
    GUIComponent::NavigationBar, 399

mValue  
    Core::Node< T >, 402  
    GUIVisualizer::Node, 419

mVisible  
    GUI::Component, 160  
    GUI::Container, 177  
    GUIComponent::Button, 85  
    GUIComponent::Card, 95  
    GUIComponent::CodeHighlighter, 150  
    GUIComponent::Footer< T >, 296  
    GUIComponent::InputField, 320  
    GUIComponent::IntegerInputField, 334  
    GUIComponent::NavigationBar, 399  
    GUIComponent::OperationContainer, 432  
    GUIComponent::OperationList, 444  
    GUIComponent::OptionInputField, 455  
    GUIComponent::StringInputField, 627  
    GUIVisualizer::CircularLinkedList, 129  
    GUIVisualizer::DoublyLinkedList, 239  
    GUIVisualizer::DynamicArray, 271  
    GUIVisualizer::LinkedList, 357  
    GUIVisualizer::Node, 420  
    GUIVisualizer::SinglyLinkedList, 531

name  
    DSInfo::Info, 307

navigation  
    State::ArrayState< T >, 70  
    State::CLLState, 139  
    State::DLLState, 207  
    State::DynamicArrayState, 281

State::HomepageState, 303  
State::LLState< T >, 386  
State::QueueState, 496  
State::SLLState, 541  
State::StackState, 581  
State::State, 587  
State::StaticArrayState, 614

NavigationBar  
    GUIComponent::NavigationBar, 391

NavigationBar\_Background  
    ColorTheme, 20

NavigationBar\_SelectedTitle  
    ColorTheme, 20

NavigationBar\_UnselectedTitle  
    ColorTheme, 20

Node  
    Core::Node< T >, 400, 401  
    GUIVisualizer::Node, 407

NonCopyable  
    NonCopyable< T >, 421

NonCopyable< T >, 420  
    ~NonCopyable, 421  
    NonCopyable, 421  
    operator=, 421

None  
    Category, 19  
    DataStructures, 22  
    States, 26

OpenFileDialog  
    Utils, 28

OperationContainer  
    GUIComponent::OperationContainer, 425

OperationList  
    GUIComponent::OperationList, 436

operationList  
    State::ArrayState< T >, 70  
    State::CLLState, 139  
    State::DLLState, 207  
    State::DynamicArrayState, 282  
    State::LLState< T >, 387  
    State::QueueState, 496  
    State::SLLState, 541  
    State::StackState, 582  
    State::StaticArrayState, 615

operator!=  
    Core::List< T >::const\_iterator, 164  
    Core::List< T >::iterator, 337

operator\*  
    Core::List< T >::const\_iterator, 164  
    Core::List< T >::iterator, 337

operator+  
    Core::List< T >::const\_iterator, 164  
    Core::List< T >::iterator, 337

operator++  
    Core::List< T >::const\_iterator, 165  
    Core::List< T >::iterator, 338

operator-  
    Core::List< T >::const\_iterator, 165

Core::List< T >::iterator, 338  
**operator->**  
 Core::List< T >::const\_iterator, 166  
 Core::List< T >::iterator, 339  
**operator--**  
 Core::List< T >::const\_iterator, 165, 166  
 Core::List< T >::iterator, 339  
**operator=**  
 Core::Deque< T >, 189, 190  
 Core::List< T >, 367, 368  
 Core::List< T >::const\_iterator, 166, 167  
 Core::List< T >::iterator, 339  
 Core::Queue< T >, 478, 479  
 Core::Stack< T >, 563  
 NonCopyable< T >, 421  
 Settings, 500  
**operator==**  
 Core::List< T >::const\_iterator, 167  
 Core::List< T >::iterator, 340  
**operator[]**  
 Core::Deque< T >, 190, 191  
 Core::List< T >, 368, 369  
 Core::Queue< T >, 479  
 Core::Stack< T >, 564  
 GUIVisualizer::DynamicArray, 264, 265  
**optionContainers**  
 GUIComponent::OperationList, 444  
**OptionInputField**  
 GUIComponent::OptionInputField, 447  
**Orientation**  
 GUIVisualizer::CircularLinkedList, 116  
 GUIVisualizer::DoublyLinkedList, 226  
 GUIVisualizer::LinkedList, 346  
 GUIVisualizer::SinglyLinkedList, 519  
 Stack.hpp, 664  
**OrientationCount**  
 GUIVisualizer::CircularLinkedList, 116  
 GUIVisualizer::DoublyLinkedList, 226  
 GUIVisualizer::LinkedList, 346  
 GUIVisualizer::SinglyLinkedList, 519  
**pack**  
 GUI::Container, 174  
 GUIComponent::Footer< T >, 293  
 GUIComponent::OperationContainer, 429  
 GUIComponent::OperationList, 440  
 GUIComponent::OptionInputField, 451  
 GUIVisualizer::CircularLinkedList, 123  
 GUIVisualizer::DoublyLinkedList, 233  
 GUIVisualizer::DynamicArray, 265  
 GUIVisualizer::LinkedList, 352  
 GUIVisualizer::SinglyLinkedList, 525  
**Pause**  
 Animation::AnimationController< T >, 43  
**Peek**  
 Algorithm::Stack, 548  
**PeekBack**  
 Algorithm::Queue, 464  
**PeekFront**  
 Algorithm::Queue, 464  
**PendingChange**  
 State::StateStack::PendingChange, 456, 457  
**PlayingAt**  
 Animation::AnimationState< T >, 51  
**pointer**  
 Core::List< T >::const\_iterator, 162  
 Core::List< T >::iterator, 336  
**Pop**  
 Algorithm::Stack, 548  
 State::StateStack, 590  
**pop**  
 Core::Queue< T >, 480  
 Core::Stack< T >, 565  
**pop\_back**  
 Core::Deque< T >, 191  
 Core::List< T >, 369  
 Core::Queue< T >, 480  
 Core::Stack< T >, 565  
**pop\_front**  
 Core::Deque< T >, 191  
 Core::List< T >, 369  
 Core::Queue< T >, 480  
 Core::Stack< T >, 565  
**PopAnimation**  
 Animation::AnimationController< T >, 43  
**PopBack**  
 Algorithm::DynamicArray, 247  
**PopState**  
 State::StateStack, 591  
**Ptr**  
 Animation::AnimationController< T >, 39  
 GUI::Component, 153  
 GUI::Container, 170  
 GUIComponent::Button, 74  
 GUIComponent::Card, 88  
 GUIComponent::CodeHighlighter, 142  
 GUIComponent::Footer< T >, 289  
 GUIComponent::InputField, 311  
 GUIComponent::IntegerInputField, 325  
 GUIComponent::NavigationBar, 390  
 GUIComponent::OperationContainer, 425  
 GUIComponent::OperationList, 435  
 GUIComponent::OptionInputField, 447  
 GUIComponent::StringInputField, 618  
 GUIVisualizer::CircularLinkedList, 115  
 GUIVisualizer::DoublyLinkedList, 225  
 GUIVisualizer::DynamicArray, 257  
 GUIVisualizer::LinkedList, 345  
 GUIVisualizer::Node, 406  
 GUIVisualizer::SinglyLinkedList, 518  
 State::ArrayState< T >, 62  
 State::CLLState, 132  
 State::DLLState, 200  
 State::DynamicArrayList, 274  
 State::HomepageState, 300  
 State::LLState< T >, 377  
 State::QueueState, 489

State::SLLState, 534  
State::StackState, 574  
State::State, 584  
State::StaticArrayState, 607  
ptr  
Core::List< T >::const\_iterator, 168  
Core::List< T >::iterator, 341  
Push  
Algorithm::DynamicArray, 247  
Algorithm::Stack, 548  
State::StateStack, 590  
push  
Core::Queue< T >, 480, 481  
Core::Stack< T >, 565  
push\_back  
Core::Deque< T >, 192  
Core::List< T >, 369  
Core::Queue< T >, 481  
Core::Stack< T >, 566  
push\_front  
Core::Deque< T >, 192  
Core::List< T >, 370  
Core::Queue< T >, 481  
Core::Stack< T >, 566  
PushBack  
Algorithm::DynamicArray, 248  
PushState  
State::StateStack, 591  
Queue  
Algorithm::Queue, 460  
DataStructures, 22  
States, 26  
Textures, 26  
queue  
State::QueueState, 497  
Queue.cpp  
ArrowType, 654  
QueueState  
State::QueueState, 490  
Rand  
Utils, 28  
Random  
Algorithm::Algorithm< GUIAlgorithm, Animation-State >, 33  
Algorithm::CircularLinkedList, 107  
Algorithm::DoublyLinkedList, 216  
Algorithm::DynamicArray, 248  
Algorithm::Queue, 464  
Algorithm::SinglyLinkedList, 509  
Algorithm::Stack, 549  
Algorithm::StaticArray, 600  
RandomFixedSize  
Algorithm::Algorithm< GUIAlgorithm, Animation-State >, 33  
Algorithm::CircularLinkedList, 107  
Algorithm::DoublyLinkedList, 216  
Algorithm::DynamicArray, 248  
Algorithm::Queue, 464  
Algorithm::SinglyLinkedList, 509  
Algorithm::Stack, 549  
Algorithm::StaticArray, 600  
Algorithm::Queue, 464  
Algorithm::SinglyLinkedList, 509  
Algorithm::Stack, 549  
Algorithm::StaticArray, 600  
RandomFixedSizeGenerator  
Algorithm::Algorithm< GUIAlgorithm, Animation-State >, 34  
Algorithm::CircularLinkedList, 107  
Algorithm::DoublyLinkedList, 217  
Algorithm::DynamicArray, 249  
Algorithm::Queue, 465  
Algorithm::SinglyLinkedList, 510  
Algorithm::Stack, 549  
Algorithm::StaticArray, 600  
RandomGenerator  
Algorithm::Algorithm< GUIAlgorithm, Animation-State >, 34  
Algorithm::CircularLinkedList, 107  
Algorithm::DoublyLinkedList, 217  
Algorithm::DynamicArray, 249  
Algorithm::Queue, 465  
Algorithm::SinglyLinkedList, 510  
Algorithm::Stack, 549  
Algorithm::StaticArray, 601  
Randomize  
GUIComponent::InputField, 315  
GUIComponent::IntegerField, 329  
GUIComponent::StringInputField, 622  
RAYGUI\_IMPLEMENTATION  
Application.cpp, 677  
ReadFromExternalFile  
Algorithm::Algorithm< GUIAlgorithm, Animation-State >, 34  
Algorithm::CircularLinkedList, 108  
Algorithm::DoublyLinkedList, 217  
Algorithm::DynamicArray, 249  
Algorithm::Queue, 465  
Algorithm::SinglyLinkedList, 510  
Algorithm::Stack, 550  
Algorithm::StaticArray, 601  
ReadFromFileGenerator  
Algorithm::Algorithm< GUIAlgorithm, Animation-State >, 35  
Algorithm::CircularLinkedList, 108  
Algorithm::DoublyLinkedList, 218  
Algorithm::DynamicArray, 250  
Algorithm::Queue, 466  
Algorithm::SinglyLinkedList, 511  
Algorithm::Stack, 550  
Algorithm::StaticArray, 601  
ReadInputFromFile  
Utils, 28  
ReCalculateEnds  
AnimationFactory, 18  
reference  
Core::List< T >::const\_iterator, 162  
Core::List< T >::iterator, 336  
Register

CategoryInfo, 98  
 DSInfo, 241  
**RegisterState**  
 State::StateStack, 592  
**RegisterStates**  
 Application, 57  
**Relayout**  
 GUIVisualizer::CircularLinkedList, 123  
 GUIVisualizer::DoublyLinkedList, 233  
 GUIVisualizer::DynamicArray, 265  
 GUIVisualizer::LinkedList, 352  
 GUIVisualizer::SinglyLinkedList, 526  
**Remove**  
 Algorithm::DynamicArray, 250  
**remove**  
 Core::Deque< T >, 192, 193  
 Core::List< T >, 370, 371  
 Core::Queue< T >, 481, 482  
 Core::Stack< T >, 566, 567  
**remove\_if**  
 Core::Deque< T >, 194  
 Core::List< T >, 371  
 Core::Queue< T >, 483  
 Core::Stack< T >, 568  
**Render**  
 Application, 57  
**RequestStackClear**  
 State::ArrayState< T >, 67  
 State::CLLState, 136  
 State::DLLState, 204  
 State::DynamicArrayState, 278  
 State::HomepageState, 301  
 State::LLState< T >, 382  
 State::QueueState, 493  
 State::SLLState, 538  
 State::StackState, 578  
 State::State, 585  
 State::StaticArrayState, 611  
**RequestStackPop**  
 State::ArrayState< T >, 67  
 State::CLLState, 136  
 State::DLLState, 204  
 State::DynamicArrayState, 278  
 State::HomepageState, 302  
 State::LLState< T >, 382  
 State::QueueState, 493  
 State::SLLState, 538  
 State::StackState, 578  
 State::State, 585  
 State::StaticArrayState, 611  
**RequestStackPush**  
 State::ArrayState< T >, 67  
 State::CLLState, 136  
 State::DLLState, 204  
 State::DynamicArrayState, 279  
 State::HomepageState, 302  
 State::LLState< T >, 383  
 State::QueueState, 494  
 State::SLLState, 538  
 State::StackState, 578  
 State::State, 585  
 State::StaticArrayState, 611  
**State::SLLState**, 539  
**State::StackState**, 579  
**State::State**, 586  
**State::StaticArrayState**, 612  
**Reserve**  
 GUIVisualizer::DynamicArray, 265  
**Reset**  
 Animation::AnimationController< T >, 43  
 Animation::AnimationState< T >, 51  
**reset**  
 Core::Deque< T >, 194  
 Core::List< T >, 372  
 Core::Queue< T >, 483  
 Core::Stack< T >, 568  
**ResetArrow**  
 GUIVisualizer::CircularLinkedList, 123  
 GUIVisualizer::DoublyLinkedList, 233  
 GUIVisualizer::SinglyLinkedList, 526  
**ResetCurrent**  
 Animation::AnimationController< T >, 44  
**ResetVisualizer**  
 Algorithm::CircularLinkedList, 108  
 Algorithm::DoublyLinkedList, 218  
 Algorithm::DynamicArray, 250  
 Algorithm::SinglyLinkedList, 511  
 Algorithm::StaticArray, 601  
**Resize**  
 GUIVisualizer::DynamicArray, 267  
**resize**  
 Core::Deque< T >, 194  
 Core::List< T >, 372  
 Core::Queue< T >, 483, 484  
 Core::Stack< T >, 568  
**reverse**  
 Core::Deque< T >, 195  
 Core::List< T >, 372  
 Core::Queue< T >, 484  
 Core::Stack< T >, 569  
**Right**  
 GUIComponent::Button, 75  
**Run**  
 Application, 57  
**RunAll**  
 Animation::AnimationController< T >, 44  
**SCREEN\_HEIGHT**  
 global, 23  
**SCREEN\_WIDTH**  
 global, 23  
**Search**  
 Algorithm::CircularLinkedList, 109  
 Algorithm::DoublyLinkedList, 218  
 Algorithm::DynamicArray, 250  
 Algorithm::SinglyLinkedList, 511  
 Algorithm::StaticArray, 602  
**select**  
 GUI::Component, 156  
 GUI::Container, 175  
 GUIComponent::Button, 79

GUIComponent::Card, 92  
    GUIComponent::CodeHighlighter, 146  
    GUIComponent::Footer< T >, 294  
    GUIComponent::InputField, 315  
    GUIComponent::IntegerInputField, 329  
    GUIComponent::NavigationBar, 394  
    GUIComponent::OperationContainer, 429  
    GUIComponent::OperationList, 440  
    GUIComponent::OptionInputField, 452  
    GUIComponent::StringInputField, 622  
    GUIVisualizer::CircularLinkedList, 123  
    GUIVisualizer::DoublyLinkedList, 233  
    GUIVisualizer::DynamicArray, 267  
    GUIVisualizer::LinkedList, 352  
    GUIVisualizer::Node, 413  
    GUIVisualizer::SinglyLinkedList, 526

SetAction  
    GUIComponent::Button, 80

SetActionDescription  
    Animation::AnimationState< T >, 52

SetActionOnHover  
    GUIComponent::Button, 80

SetActive  
    GUIVisualizer::Node, 413

SetActiveTitle  
    GUIComponent::NavigationBar, 395

SetAnimation  
    Animation::AnimationController< T >, 44  
    Animation::AnimationState< T >, 52

SetArrowType  
    GUIVisualizer::CircularLinkedList, 124  
    GUIVisualizer::SinglyLinkedList, 526

SetArrowTypeNext  
    GUIVisualizer::DoublyLinkedList, 234

SetArrowTypePrev  
    GUIVisualizer::DoublyLinkedList, 234

SetButtonColor  
    GUIComponent::Button, 80

SetButtonHoverColor  
    GUIComponent::Button, 80

SetCategory  
    GUIComponent::NavigationBar, 395

SetCircularArrowType  
    GUIVisualizer::CircularLinkedList, 124

SetCircularEnds  
    GUIVisualizer::CircularLinkedList, 124

SetConstraint  
    GUIComponent::IntegerInputField, 330

SetCurrentAction  
    State::ArrayState< T >, 68  
    State::CLLState, 137  
    State::DLLState, 205  
    State::DynamicArrayState, 279  
    State::LLState< T >, 383  
    State::QueueState, 494  
    State::SLLState, 539  
    State::StackState, 579  
    State::StaticArrayState, 612

SetCurrentError  
    State::ArrayState< T >, 68  
    State::CLLState, 137  
    State::DLLState, 205  
    State::DynamicArrayState, 279  
    State::LLState< T >, 383  
    State::QueueState, 494  
    State::SLLState, 539  
    State::StackState, 579  
    State::StaticArrayState, 612

SetDirectLink  
    GUIComponent::NavigationBar, 395

SetDuration  
    Animation::AnimationState< T >, 52

SetEditMode  
    GUIComponent::InputField, 316  
    GUIComponent::IntegerInputField, 330  
    GUIComponent::StringInputField, 623

SetFontSize  
    GUIComponent::Button, 80

SetHighlightLine  
    Animation::AnimationState< T >, 53

SetHomePageID  
    GUIComponent::NavigationBar, 395

SetInputFieldSize  
    GUIComponent::InputField, 316  
    GUIComponent::IntegerInputField, 330  
    GUIComponent::StringInputField, 623

SetLabel  
    GUIComponent::InputField, 316  
    GUIComponent::IntegerInputField, 330  
    GUIComponent::StringInputField, 623  
    GUIVisualizer::Node, 414

SetLabelFontSize  
    GUIVisualizer::Node, 414

SetLabelSize  
    GUIComponent::InputField, 316  
    GUIComponent::IntegerInputField, 330  
    GUIComponent::StringInputField, 623

SetLink  
    GUIComponent::Card, 92

SetNodeState  
    GUIVisualizer::Node, 414

SetNoFieldOption  
    GUIComponent::OptionInputField, 452

SetOption  
    GUIComponent::OptionInputField, 452

SetOrientation  
    GUIVisualizer::CircularLinkedList, 124  
    GUIVisualizer::DoublyLinkedList, 234  
    GUIVisualizer::LinkedList, 352  
    GUIVisualizer::SinglyLinkedList, 526

SetPosition  
    GUI::Component, 157  
    GUI::Container, 175  
    GUIComponent::Button, 80, 81  
    GUIComponent::Card, 92, 93  
    GUIComponent::CodeHighlighter, 147

GUIComponent::Footer< T >, 294  
 GUIComponent::InputField, 316, 317  
 GUIComponent::IntegerField, 330, 331  
 GUIComponent::NavigationBar, 395, 396  
 GUIComponent::OperationContainer, 429, 430  
 GUIComponent::OperationList, 440, 441  
 GUIComponent::OptionInputField, 452, 453  
 GUIComponent::StringInputField, 623, 624  
 GUIVisualizer::CircularLinkedList, 124, 125  
 GUIVisualizer::DoublyLinkedList, 234, 235  
 GUIVisualizer::DynamicArray, 267  
 GUIVisualizer::LinkedList, 353  
 GUIVisualizer::Node, 414, 415  
 GUIVisualizer::SinglyLinkedList, 527  
**SetRadius**  
 GUIVisualizer::Node, 415  
**SetReachable**  
 GUIVisualizer::Node, 415  
**SetShape**  
 GUIVisualizer::CircularLinkedList, 125  
 GUIVisualizer::DoublyLinkedList, 235  
 GUIVisualizer::DynamicArray, 268  
 GUIVisualizer::LinkedList, 353  
 GUIVisualizer::Node, 415  
 GUIVisualizer::SinglyLinkedList, 527  
**ShowAction**  
 GUIComponent::CodeHighlighter, 147  
**ShowCode**  
 GUIComponent::CodeHighlighter, 147  
**ShowHeadAndTail**  
 GUIVisualizer::CircularLinkedList, 125  
 GUIVisualizer::DoublyLinkedList, 235  
 GUIVisualizer::LinkedList, 354  
 GUIVisualizer::SinglyLinkedList, 527  
**SetSize**  
 GUIComponent::Button, 81  
**SetSourceDataStructure**  
 Animation::AnimationState< T >, 53  
**SetSpeed**  
 Animation::AnimationController< T >, 44  
**SetStateID**  
 GUIComponent::Card, 93  
**SetText**  
 GUIComponent::Button, 81  
 GUIComponent::Card, 93  
**SetTextAlignment**  
 GUIComponent::Button, 81  
**SetTextColor**  
 GUIComponent::Button, 81  
**Settings**, 497  
 ~Settings, 499  
 getColor, 499  
 getInstance, 500  
 Load, 500  
 LoadDarkColors, 500  
 LoadDefaultColors, 500  
 mColors, 501  
 mCurrentColorTheme, 501  
 operator=, 500  
 Settings, 498, 499  
 SwitchTheme, 501  
**SetValue**  
 GUIVisualizer::Node, 415  
**SetValueFontSize**  
 GUIVisualizer::Node, 416  
**SetVisibleTitle**  
 GUIComponent::NavigationBar, 396  
**SetVisible**  
 GUI::Component, 157  
 GUI::Container, 176  
 GUIComponent::Button, 82  
 GUIComponent::Card, 93  
 GUIComponent::CodeHighlighter, 148  
 GUIComponent::Footer< T >, 295  
 GUIComponent::InputField, 317  
 GUIComponent::IntegerField, 331  
 GUIComponent::NavigationBar, 396  
 GUIComponent::OperationContainer, 430  
 GUIComponent::OperationList, 441  
 GUIComponent::OptionInputField, 453  
 GUIComponent::StringInputField, 624  
 GUIVisualizer::CircularLinkedList, 125  
 GUIVisualizer::DoublyLinkedList, 235  
 GUIVisualizer::DynamicArray, 268  
 GUIVisualizer::LinkedList, 354  
 GUIVisualizer::Node, 416  
 GUIVisualizer::SinglyLinkedList, 528  
**Shape**  
 GUIVisualizer::Node, 406  
**ShapeCount**  
 GUIVisualizer::Node, 407  
**ShowOptions**  
 GUIComponent::OperationList, 441  
**Silkscreen**  
 Fonts, 22  
**SinglyLinkedList**  
 Algorithm::SinglyLinkedList, 504, 505  
 DataStructures, 22  
 GUIVisualizer::SinglyLinkedList, 519  
 States, 26  
 Textures, 26  
**SinglyLinkedList.cpp**  
 ArrowType, 658  
**size**  
 Algorithm::CircularLinkedList, 109  
 Algorithm::DoublyLinkedList, 218  
 Algorithm::DynamicArray, 251  
 Algorithm::Queue, 466  
 Algorithm::SinglyLinkedList, 511  
 Algorithm::Stack, 550  
 Algorithm::StaticArray, 602  
 Core::Deque< T >, 195  
 Core::List< T >, 373  
 Core::Queue< T >, 484  
 Core::Stack< T >, 569  
 GUIVisualizer::CircularLinkedList, 126

GUIVisualizer::DoublyLinkedList, 236  
    GUIVisualizer::LinkedList, 354  
    GUIVisualizer::SinglyLinkedList, 528

Skip

- GUIVisualizer::CircularLinkedList, 116
- GUIVisualizer::DoublyLinkedList, 226
- GUIVisualizer::LinkedList, 345
- GUIVisualizer::SinglyLinkedList, 519

SLL

- State::SLLState, 542

SLLAnimation

- AnimationState.hpp, 674

SLLAnimationController

- AnimationController.hpp, 667

SLLState

- State::SLLState, 535

Square

- GUIVisualizer::Node, 407

src/Algorithms/Algorithm.hpp, 635, 636  
src/Algorithms/Array/DynamicArray.cpp, 639  
src/Algorithms/Array/DynamicArray.hpp, 640, 641  
src/Algorithms/Array/StaticArray.cpp, 643  
src/Algorithms/Array/StaticArray.hpp, 643, 644  
src/Algorithms/LinkedList/CircularLinkedList.cpp, 645  
src/Algorithms/LinkedList/CircularLinkedList.hpp, 646, 647  
src/Algorithms/LinkedList/DoublyLinkedList.cpp, 649  
src/Algorithms/LinkedList/DoublyLinkedList.hpp, 651, 652  
src/Algorithms/LinkedList/Queue.cpp, 654  
src/Algorithms/LinkedList/Queue.hpp, 655, 656  
src/Algorithms/LinkedList/SinglyLinkedList.cpp, 657  
src/Algorithms/LinkedList/SinglyLinkedList.hpp, 659, 660  
src/Algorithms/LinkedList/Stack.cpp, 662  
src/Algorithms/LinkedList/Stack.hpp, 662, 664  
src/Animation/AnimationController.hpp, 666, 667  
src/Animation/AnimationFactory.cpp, 670  
src/Animation/AnimationFactory.hpp, 671, 672  
src/Animation/AnimationState.hpp, 672, 674  
src/Application.cpp, 676  
src/Application.hpp, 677, 678  
src/Component.cpp, 679  
src/Component.hpp, 679, 680  
src/Components/Common/Button.cpp, 681  
src/Components/Common/Button.hpp, 681, 682  
src/Components/Common/Card.cpp, 683  
src/Components/Common/Card.hpp, 683, 684  
src/Components/Common/CodeHighlighter.cpp, 685  
src/Components/Common/CodeHighlighter.hpp, 685, 686  
src/Components/Common/Footer.hpp, 687  
src/Components/Common/InputField.cpp, 689  
src/Components/Common/InputField.hpp, 689, 690  
src/Components/Common/IntegerField.cpp, 691  
src/Components/Common/IntegerField.hpp, 691, 692  
src/Components/Common/NavigationBar.cpp, 692

    src/Components/Common/NavigationBar.hpp, 692, 693  
    src/Components/Common/OperationContainer.cpp, 694  
    src/Components/Common/OperationContainer.hpp, 694, 695  
    src/Components/Common/OperationList.cpp, 696  
    src/Components/Common/OperationList.hpp, 696, 697  
    src/Components/Common/OptionInputField.cpp, 697  
    src/Components/Common/OptionInputField.hpp, 698  
    src/Components/Common/StringInputField.cpp, 699  
    src/Components/Common/StringInputField.hpp, 700  
    src/Components/Visualization/CircularLinkedList.cpp, 646  
    src/Components/Visualization/CircularLinkedList.hpp, 648, 649  
    src/Components/Visualization/DoublyLinkedList.cpp, 650  
    src/Components/Visualization/DoublyLinkedList.hpp, 653  
    src/Components/Visualization/DynamicArray.cpp, 639  
    src/Components/Visualization/DynamicArray.hpp, 641, 642  
    src/Components/Visualization/LinkedList.cpp, 701  
    src/Components/Visualization/LinkedList.hpp, 701, 702  
    src/Components/Visualization/Node.cpp, 703  
    src/Components/Visualization/Node.hpp, 703, 704  
    src/Components/Visualization/SinglyLinkedList.cpp, 658  
    src/Components/Visualization/SinglyLinkedList.hpp, 661  
    src/Container.cpp, 706  
    src/Container.hpp, 707  
    src/Core/Deque.hpp, 708, 709  
    src/Core/List.hpp, 709, 710  
    src/Core/Node.hpp, 705, 706  
    src/Core/Queue.hpp, 656, 657  
    src/Core/Stack.hpp, 665  
    src/FontHolder.cpp, 717  
    src/FontHolder.hpp, 717, 718  
    src/Global.hpp, 719  
    src/Identifiers/CategoryIdentifiers.hpp, 720  
    src/Identifiers/CategoryInfo.cpp, 720  
    src/Identifiers/CategoryInfo.hpp, 721  
    src/Identifiers/ColorThemelIdentifiers.hpp, 722, 723  
    src/Identifiers/DSIdentifiers.hpp, 724, 725  
    src/Identifiers/DSInfo.cpp, 725  
    src/Identifiers/DSInfo.hpp, 725, 726  
    src/Identifiers/ResourceIdentifiers.hpp, 727  
    src/Identifiers/StatelIdentifiers.hpp, 728  
    src/Main.cpp, 729  
    src/NonCopyable.hpp, 729, 730  
    src/Settings.cpp, 730  
    src/Settings.hpp, 730, 731  
    src/State.cpp, 731  
    src/State.hpp, 732, 733  
    src/States/Array/ArrayState.hpp, 733, 734  
    src/States/Array/DynamicArrayState.cpp, 738  
    src/States/Array/DynamicArrayState.hpp, 738, 739  
    src/States/Array/StaticArrayState.cpp, 740  
    src/States/Array/StaticArrayState.hpp, 740, 741

src/States/HomepageState.cpp, 742  
 src/States/HomepageState.hpp, 742, 743  
 src/States/LinkedList/CLLState.cpp, 743  
 src/States/LinkedList/CLLState.hpp, 744  
 src/States/LinkedList/DLLState.cpp, 745  
 src/States/LinkedList/DLLState.hpp, 745, 746  
 src/States/LinkedList/LLState.hpp, 747, 748  
 src/States/LinkedList/QueueState.cpp, 751  
 src/States/LinkedList/QueueState.hpp, 751, 752  
 src/States/LinkedList/SLLState.cpp, 753  
 src/States/LinkedList/SLLState.hpp, 753, 754  
 src/States/LinkedList/StackState.cpp, 755  
 src/States/LinkedList/StackState.hpp, 755, 756  
 src/StateStack.cpp, 756  
 src/StateStack.hpp, 757, 758  
 src/TextureHolder.cpp, 759  
 src/TextureHolder.hpp, 759, 760  
 src/Utils/Utils.cpp, 760  
 src/Utils/Utils.hpp, 761, 762  
**Stack**  
 Algorithm::Stack, 545  
 DataStructures, 22  
 States, 26  
 Textures, 26  
**Stack.hpp**  
 ArrowType, 664  
 Orientation, 664  
**StackState**  
 State::StackState, 575  
**State**, 25  
 GUIVisualizer::Node, 407  
 State::State, 584  
**State::ArrayState< T >**, 60  
 ~ArrayState, 63  
 activeDS, 69  
 AddAccessOperation, 63  
 AddDeleteOperation, 63  
 AddInitializeOperation, 64  
 AddInsertOperation, 64  
 AddIntFieldOperationOption, 64  
 AddNoFieldOperationOption, 64  
 AddOperations, 64  
 AddSearchOperation, 65  
 AddStringFieldOption, 65  
 AddUpdateOperation, 65  
 animController, 69  
 ArrayState, 63  
 ClearAction, 65  
 ClearError, 66  
 codeHighlighter, 69  
 Draw, 66  
 DrawCurrentActionText, 66  
 DrawCurrentErrorText, 66  
 footer, 69  
 GetContext, 66  
 InitNavigationBar, 67  
 mContext, 70  
 mCurrentAction, 70  
 mCurrentError, 70  
 mStack, 70  
 navigation, 70  
 operationList, 70  
 Ptr, 62  
 RequestStackClear, 67  
 RequestStackPop, 67  
 RequestStackPush, 67  
 SetCurrentAction, 68  
 SetCurrentError, 68  
 Success, 68  
 Update, 68  
**State::ArrayState< T >::IntegerInput**, 320  
 label, 320  
 maxValue, 320  
 minValue, 321  
 width, 321  
**State::CLLState**, 130  
 ~CLLState, 133  
 activeDS, 138  
 AddDeleteOperation, 133  
 AddInitializeOperation, 133  
 AddInsertOperation, 133  
 AddIntFieldOperationOption, 133  
 AddNoFieldOperationOption, 134  
 AddOperations, 134  
 AddSearchOperation, 134  
 AddStringFieldOption, 134  
 AddUpdateOperation, 134  
 animController, 138  
 ClearAction, 135  
 ClearError, 135  
 CLL, 138  
 CLLState, 132  
 codeHighlighter, 138  
 Draw, 135  
 DrawCurrentActionText, 135  
 DrawCurrentErrorText, 135  
 footer, 138  
 GetContext, 135  
 InitNavigationBar, 136  
 mContext, 138  
 mCurrentAction, 139  
 mCurrentError, 139  
 mStack, 139  
 navigation, 139  
 operationList, 139  
 Ptr, 132  
 RequestStackClear, 136  
 RequestStackPop, 136  
 RequestStackPush, 136  
 SetCurrentAction, 137  
 SetCurrentError, 137  
 Success, 137  
 Update, 137  
**State::DLLState**, 197  
 ~DLLState, 201  
 activeDS, 206

AddDeleteOperation, 201  
AddInitializeOperation, 201  
AddInsertOperation, 201  
AddIntFieldOperationOption, 201  
AddNoFieldOperationOption, 202  
AddOperations, 202  
AddSearchOperation, 202  
AddStringFieldOption, 202  
AddUpdateOperation, 202  
animController, 206  
ClearAction, 203  
ClearError, 203  
codeHighlighter, 206  
DLLState, 200  
Draw, 203  
DrawCurrentActionText, 203  
DrawCurrentErrorText, 203  
footer, 206  
GetContext, 203  
InitNavigationBar, 204  
mContext, 206  
mCurrentAction, 206  
mCurrentError, 207  
mDLL, 207  
mStack, 207  
navigation, 207  
operationList, 207  
Ptr, 200  
RequestStackClear, 204  
RequestStackPop, 204  
RequestStackPush, 204  
SetCurrentAction, 205  
SetCurrentError, 205  
Success, 205  
Update, 205  
State::DynamicArrayState, 271  
~DynamicArrayState, 275  
activeDS, 280  
AddAccessOperation, 275  
AddDeleteOperation, 275  
AddInitializeOperation, 275  
AddInsertOperation, 275  
AddIntFieldOperationOption, 276  
AddNoFieldOperationOption, 276  
AddOperations, 276  
AddSearchOperation, 276  
AddStringFieldOption, 276  
AddUpdateOperation, 277  
animController, 280  
ClearAction, 277  
ClearError, 277  
codeHighlighter, 280  
Draw, 277  
DrawCurrentActionText, 277  
DrawCurrentErrorText, 278  
DynamicArrayState, 274  
footer, 281  
GetContext, 278  
InitNavigationBar, 278  
mContext, 281  
mCurrentAction, 281  
mCurrentError, 281  
mDynamicArray, 281  
mStack, 281  
navigation, 281  
operationList, 282  
Ptr, 274  
RequestStackClear, 278  
RequestStackPop, 278  
RequestStackPush, 279  
SetCurrentAction, 279  
SetCurrentError, 279  
Success, 280  
Update, 280  
State::HomepageState, 297  
~HomepageState, 300  
CreateCard, 300  
Draw, 300  
DrawIntroduction, 301  
GetContext, 301  
hasInitializeCard, 303  
HomepageState, 300  
InitCards, 301  
InitNavigationBar, 301  
mCards, 303  
mContext, 303  
mStack, 303  
navigation, 303  
Ptr, 300  
RequestStackClear, 301  
RequestStackPop, 302  
RequestStackPush, 302  
Update, 302  
State::LLState< T >, 375  
~LLState, 379  
activeDS, 385  
AddDeleteOperation, 379  
AddInitializeOperation, 379  
AddInsertOperation, 379  
AddIntFieldOperationOption, 379  
AddNoFieldOperationOption, 380  
AddOperations, 380  
AddSearchOperation, 380  
AddStringFieldOption, 380  
AddUpdateOperation, 381  
animController, 385  
ClearAction, 381  
ClearError, 381  
codeHighlighter, 386  
Draw, 381  
DrawCurrentActionText, 381  
DrawCurrentErrorText, 382  
footer, 386  
GetContext, 382  
InitNavigationBar, 382  
LLState, 377

mContext, 386  
 mCurrentAction, 386  
 mCurrentError, 386  
 mStack, 386  
 navigation, 386  
 operationList, 387  
 Ptr, 377  
 RequestStackClear, 382  
 RequestStackPop, 382  
 RequestStackPush, 383  
 SetCurrentAction, 383  
 SetCurrentError, 383  
 Success, 385  
 Update, 385  
**State::LLState< T >::IntegerInput**, 321  
 label, 321  
 maxValue, 321  
 minValue, 322  
 width, 322  
**State::QueueState**, 487  
 ~QueueState, 490  
 activeDS, 495  
 AddDeleteOperation, 490  
 AddInitializeOperation, 490  
 AddInsertOperation, 490  
 AddIntFieldOperationOption, 491  
 AddNoFieldOperationOption, 491  
 AddOperations, 491  
 AddSearchOperation, 491  
 AddStringFieldOption, 491  
 AddUpdateOperation, 492  
 animController, 495  
 ClearAction, 492  
 ClearError, 492  
 codeHighlighter, 495  
 Draw, 492  
 DrawCurrentActionText, 492  
 DrawCurrentErrorText, 493  
 footer, 496  
 GetContext, 493  
 InitNavigationBar, 493  
 mContext, 496  
 mCurrentAction, 496  
 mCurrentError, 496  
 mStack, 496  
 navigation, 496  
 operationList, 496  
 Ptr, 489  
 queue, 497  
 QueueState, 490  
 RequestStackClear, 493  
 RequestStackPop, 493  
 RequestStackPush, 494  
 SetCurrentAction, 494  
 SetCurrentError, 494  
 Success, 495  
 Update, 495  
**State::SLLState**, 532  
 ~SLLState, 535  
 activeDS, 540  
 AddDeleteOperation, 535  
 AddInitializeOperation, 535  
 AddInsertOperation, 535  
 AddIntFieldOperationOption, 536  
 AddNoFieldOperationOption, 536  
 AddOperations, 536  
 AddSearchOperation, 536  
 AddStringFieldOption, 536  
 AddUpdateOperation, 537  
 animController, 540  
 ClearAction, 537  
 ClearError, 537  
 codeHighlighter, 540  
 Draw, 537  
 DrawCurrentActionText, 537  
 DrawCurrentErrorText, 538  
 footer, 541  
 GetContext, 538  
 InitNavigationBar, 538  
 mContext, 541  
 mCurrentAction, 541  
 mCurrentError, 541  
 mStack, 541  
 navigation, 541  
 operationList, 541  
 Ptr, 534  
 RequestStackClear, 538  
 RequestStackPop, 538  
 RequestStackPush, 539  
 SetCurrentAction, 539  
 SetCurrentError, 539  
 SLL, 542  
 SLLState, 535  
 Success, 540  
 Update, 540  
**State::StackState**, 572  
 ~StackState, 575  
 activeDS, 580  
 AddDeleteOperation, 575  
 AddInitializeOperation, 575  
 AddInsertOperation, 575  
 AddIntFieldOperationOption, 576  
 AddNoFieldOperationOption, 576  
 AddOperations, 576  
 AddSearchOperation, 576  
 AddStringFieldOption, 576  
 AddUpdateOperation, 577  
 animController, 580  
 ClearAction, 577  
 ClearError, 577  
 codeHighlighter, 580  
 Draw, 577  
 DrawCurrentActionText, 577  
 DrawCurrentErrorText, 578  
 footer, 581  
 GetContext, 578

InitNavigationBar, 578  
mContext, 581  
mCurrentAction, 581  
mCurrentError, 581  
mStack, 581  
mStackAlgorithm, 581  
navigation, 581  
operationList, 582  
Ptr, 574  
RequestStackClear, 578  
RequestStackPop, 578  
RequestStackPush, 579  
SetCurrentAction, 579  
SetCurrentError, 579  
StackState, 575  
Success, 580  
Update, 580  
State::State, 582  
~State, 584  
Draw, 584  
GetContext, 585  
InitNavigationBar, 585  
mContext, 587  
mStack, 587  
navigation, 587  
Ptr, 584  
RequestStackClear, 585  
RequestStackPop, 585  
RequestStackPush, 586  
State, 584  
Update, 586  
State::State::Context, 178  
categories, 179  
Context, 178  
dsInfo, 179  
fonts, 179  
textures, 179  
State::StateStack, 587  
Action, 589  
ApplyPendingChanges, 590  
Clear, 590  
ClearStates, 590  
createState, 590  
Draw, 591  
IsEmpty, 591  
mContext, 592  
mFactories, 592  
mPendingList, 593  
mStack, 593  
Pop, 590  
PopState, 591  
Push, 590  
PushState, 591  
RegisterState, 592  
StateStack, 590  
Update, 592  
State::StateStack::PendingChange, 456  
action, 457  
PendingChange, 456, 457  
stateID, 457  
State::StaticArrayState, 604  
~StaticArrayState, 608  
activeDS, 613  
AddAccessOperation, 608  
AddDeleteOperation, 608  
AddInitializeOperation, 608  
AddInsertOperation, 608  
AddIntFieldOperationOption, 609  
AddNoFieldOperationOption, 609  
AddOperations, 609  
AddSearchOperation, 609  
AddStringFieldOption, 609  
AddUpdateOperation, 610  
animController, 613  
ClearAction, 610  
ClearError, 610  
codeHighlighter, 613  
Draw, 610  
DrawCurrentActionText, 610  
DrawCurrentErrorText, 611  
footer, 614  
GetContext, 611  
InitNavigationBar, 611  
mContext, 614  
mCurrentAction, 614  
mCurrentError, 614  
mStack, 614  
mStaticArray, 614  
navigation, 614  
operationList, 615  
Ptr, 607  
RequestStackClear, 611  
RequestStackPop, 611  
RequestStackPush, 612  
SetCurrentAction, 612  
SetCurrentError, 612  
StaticArrayState, 607  
Success, 613  
Update, 613  
StateCount  
  GUIVisualizer::Node, 407  
stateID  
  DSInfo::Info, 308  
  GUIComponent::Card, 95  
  GUIComponent::NavigationBar::TitleInfo, 633  
  State::StateStack::PendingChange, 457  
States, 25  
  CircularLinkedList, 26  
  Count, 26  
  DoublyLinkedList, 26  
  DynamicArray, 26  
  Homepage, 26  
  ID, 25  
  None, 26  
  Queue, 26  
  SinglyLinkedList, 26

Stack, 26  
 StaticArray, 26  
 StateStack  
     State::StateStack, 590  
 StaticArray  
     Algorithm::StaticArray, 596  
     DataStructures, 22  
     States, 26  
     Textures, 26  
 StaticArrayState  
     State::StaticArrayState, 607  
 StepBackward  
     Animation::AnimationController< T >, 45  
 StepForward  
     Animation::AnimationController< T >, 45  
 stopDuration  
     Animation::AnimationController< T >, 47  
 StringInputField  
     GUIComponent::StringInputField, 618  
 Success  
     State::ArrayState< T >, 68  
     State::CLLState, 137  
     State::DLLState, 205  
     State::DynamicArrayState, 280  
     State::LLState< T >, 385  
     State::QueueState, 495  
     State::SLLState, 540  
     State::StackState, 580  
     State::StaticArrayState, 613  
 swap  
     Core::Deque< T >, 195  
     Core::List< T >, 373  
     Core::List< T >::const\_iterator, 167  
     Core::List< T >::iterator, 340  
     Core::Queue< T >, 484, 485  
     Core::Stack< T >, 569, 570  
 SwitchTheme  
     Settings, 501  
  
 Text  
     ColorTheme, 20  
 TextAlignment  
     GUIComponent::Button, 74  
 textColorTheme  
     GUIComponent::Button, 85  
 TextureHolder, 628  
     ~TextureHolder, 629  
     Get, 629, 630  
     InsertResource, 630  
     Load, 631  
     LoadFromImage, 632  
     mTextureMap, 633  
     TextureHolder, 629  
 Textures, 26  
     Blank, 26  
     CircularLinkedList, 26  
     Count, 26  
     DoublyLinkedList, 26  
     DynamicArray, 26  
     Favicon, 26  
     ID, 26  
     Queue, 26  
     SinglyLinkedList, 26  
     Stack, 26  
     StaticArray, 26  
 textures  
     State::State::Context, 179  
 thumbnail  
     DSInfo::Info, 308  
     GUIComponent::Card, 95  
 title  
     GUIComponent::Card, 96  
 titleName  
     GUIComponent::NavigationBar::TitleInfo, 633  
 toggleButton  
     GUIComponent::OperationList, 444  
 ToggleInputFields  
     GUIComponent::OptionInputField, 453  
 ToggleOperations  
     GUIComponent::OperationList, 441  
 ToggleShowAction  
     GUIComponent::CodeHighlighter, 148  
 ToggleShowCode  
     GUIComponent::CodeHighlighter, 148  
 ToggleVisible  
     GUI::Component, 159  
     GUI::Container, 176  
     GUIComponent::Button, 82  
     GUIComponent::Card, 93  
     GUIComponent::CodeHighlighter, 148  
     GUIComponent::Footer< T >, 295  
     GUIComponent::InputField, 317  
     GUIComponent::IntegerField, 331  
     GUIComponent::NavigationBar, 396  
     GUIComponent::OperationContainer, 430  
     GUIComponent::OperationList, 442  
     GUIComponent::OptionInputField, 453  
     GUIComponent::StringInputField, 624  
     GUIVisualizer::CircularLinkedList, 126  
     GUIVisualizer::DoublyLinkedList, 236  
     GUIVisualizer::DynamicArray, 268  
     GUIVisualizer::LinkedList, 354  
     GUIVisualizer::Node, 416  
     GUIVisualizer::SinglyLinkedList, 528  
 toLink  
     GUIComponent::Card, 96  
     GUIComponent::NavigationBar, 399  
 top  
     Core::Stack< T >, 570  
  
 unique  
     Core::Deque< T >, 196  
     Core::List< T >, 373  
     Core::Queue< T >, 485  
     Core::Stack< T >, 570  
 UnpackAll  
     GUI::Container, 176  
     GUIComponent::Footer< T >, 295

GUIComponent::OperationContainer, 430  
    GUIComponent::OperationList, 442  
    GUIComponent::OptionInputField, 453  
    GUIVisualizer::CircularLinkedList, 126  
    GUIVisualizer::DoublyLinkedList, 236  
    GUIVisualizer::DynamicArray, 269  
    GUIVisualizer::LinkedList, 354  
    GUIVisualizer::SinglyLinkedList, 528

Update  
    Algorithm::CircularLinkedList, 109  
    Algorithm::DoublyLinkedList, 219  
    Algorithm::DynamicArray, 251  
    Algorithm::SinglyLinkedList, 512  
    Algorithm::StaticArray, 602  
Animation::AnimationController< T >, 45  
Animation::AnimationState< T >, 53  
Application, 57  
State::ArrayState< T >, 68  
State::CLLState, 137  
State::DLLState, 205  
State::DynamicArrayState, 280  
State::HomepageState, 302  
State::LLState< T >, 385  
State::QueueState, 495  
State::SLLState, 540  
State::StackState, 580  
State::State, 586  
State::StateStack, 592  
State::StaticArrayState, 613

UpdateMouseCursorWhenHover  
    GUI::Component, 159  
    GUI::Container, 176, 177  
    GUIComponent::Button, 82  
    GUIComponent::Card, 94  
    GUIComponent::CodeHighlighter, 148, 149  
    GUIComponent::Footer< T >, 295, 296  
    GUIComponent::InputField, 317, 318  
    GUIComponent::IntegerInputField, 331, 332  
    GUIComponent::NavigationBar, 397  
    GUIComponent::OperationContainer, 431  
    GUIComponent::OperationList, 442  
    GUIComponent::OptionInputField, 454  
    GUIComponent::StringInputField, 624, 625  
    GUIVisualizer::CircularLinkedList, 126, 127  
    GUIVisualizer::DoublyLinkedList, 236, 237  
    GUIVisualizer::DynamicArray, 269  
    GUIVisualizer::LinkedList, 355  
    GUIVisualizer::Node, 416, 417  
    GUIVisualizer::SinglyLinkedList, 529

UpdatePosition  
    GUIComponent::OperationContainer, 431

UserDefined  
    Algorithm::Algorithm< GUIAlgorithm, Animation-  
        State >, 35  
    Algorithm::CircularLinkedList, 109  
    Algorithm::DoublyLinkedList, 219  
    Algorithm::DynamicArray, 251  
    Algorithm::Queue, 466

    Algorithm::SinglyLinkedList, 512  
    Algorithm::Stack, 551  
    Algorithm::StaticArray, 603

UserDefinedGenerator  
    Algorithm::Algorithm< GUIAlgorithm, Animation-  
        State >, 35  
    Algorithm::CircularLinkedList, 110  
    Algorithm::DoublyLinkedList, 219  
    Algorithm::DynamicArray, 252  
    Algorithm::Queue, 466  
    Algorithm::SinglyLinkedList, 512  
    Algorithm::Stack, 551  
    Algorithm::StaticArray, 603

Utils, 27  
    DrawIcon, 27  
    DrawTextBoxed, 27  
    DrawTextBoxedSelectable, 27  
    OpenFileDialog, 28  
    Rand, 28  
    ReadInputFromFile, 28

value\_type  
    Core::List< T >::const\_iterator, 162  
    Core::List< T >::iterator, 336

valueFontSize  
    GUIVisualizer::Node, 420

Vertical  
    GUIVisualizer::CircularLinkedList, 116  
    GUIVisualizer::DoublyLinkedList, 226  
    GUIVisualizer::LinkedList, 346  
    GUIVisualizer::SinglyLinkedList, 519

visualizer  
    Algorithm::Algorithm< GUIAlgorithm, Animation-  
        State >, 36  
    Algorithm::CircularLinkedList, 111  
    Algorithm::DoublyLinkedList, 220  
    Algorithm::DynamicArray, 253  
    Algorithm::Queue, 467  
    Algorithm::SinglyLinkedList, 513  
    Algorithm::Stack, 552  
    Algorithm::StaticArray, 604

Visualizer\_ActionText  
    ColorTheme, 20

Visualizer\_Arrow\_Active  
    ColorTheme, 21

Visualizer\_Arrow\_Default  
    ColorTheme, 21

Visualizer\_ErrorText  
    ColorTheme, 20

Visualizer\_Label  
    ColorTheme, 20

Visualizer\_Node\_Active\_Background1  
    ColorTheme, 20

Visualizer\_Node\_Active\_Background2  
    ColorTheme, 20

Visualizer\_Node\_Active\_Outline1  
    ColorTheme, 20

Visualizer\_Node\_Active\_Outline2  
    ColorTheme, 20

Visualizer\_Node\_Active\_Text1  
    ColorTheme, 20

Visualizer\_Node\_Active\_Text2  
    ColorTheme, 20

Visualizer\_Node\_ActiveBlue\_Background1  
    ColorTheme, 20

Visualizer\_Node\_ActiveBlue\_Background2  
    ColorTheme, 20

Visualizer\_Node\_ActiveBlue\_Outline1  
    ColorTheme, 20

Visualizer\_Node\_ActiveBlue\_Outline2  
    ColorTheme, 20

Visualizer\_Node\_ActiveBlue\_Text1  
    ColorTheme, 20

Visualizer\_Node\_ActiveBlue\_Text2  
    ColorTheme, 20

Visualizer\_Node\_ActiveGreen\_Background1  
    ColorTheme, 21

Visualizer\_Node\_ActiveGreen\_Background2  
    ColorTheme, 21

Visualizer\_Node\_ActiveGreen\_Outline1  
    ColorTheme, 20

Visualizer\_Node\_ActiveGreen\_Outline2  
    ColorTheme, 21

Visualizer\_Node\_ActiveGreen\_Text1  
    ColorTheme, 21

Visualizer\_Node\_ActiveGreen\_Text2  
    ColorTheme, 21

Visualizer\_Node\_ActiveRed\_Background1  
    ColorTheme, 21

Visualizer\_Node\_ActiveRed\_Background2  
    ColorTheme, 21

Visualizer\_Node\_ActiveRed\_Outline1  
    ColorTheme, 21

Visualizer\_Node\_ActiveRed\_Outline2  
    ColorTheme, 21

Visualizer\_Node\_ActiveRed\_Text1  
    ColorTheme, 21

Visualizer\_Node\_ActiveRed\_Text2  
    ColorTheme, 21

Visualizer\_Node\_Default\_Background1  
    ColorTheme, 20

Visualizer\_Node\_Default\_Background2  
    ColorTheme, 20

Visualizer\_Node\_Default\_Outline1  
    ColorTheme, 20

Visualizer\_Node\_Default\_Outline2  
    ColorTheme, 20

Visualizer\_Node\_Default\_Text1  
    ColorTheme, 20

Visualizer\_Node\_Default\_Text2  
    ColorTheme, 20

Visualizer\_Node\_Iterated\_Background1  
    ColorTheme, 21

Visualizer\_Node\_Iterated\_Background2  
    ColorTheme, 21

Visualizer\_Node\_Iterated\_Outline1  
    ColorTheme, 21

Visualizer\_Node\_Iterated\_Outline2  
    ColorTheme, 21

Visualizer\_Node\_Iterated\_Text1  
    ColorTheme, 21

Visualizer\_Node\_Iterated\_Text2  
    ColorTheme, 21

width  
    State::ArrayState< T >::IntegerInput, 321  
    State::LLState< T >::IntegerInput, 322

WindowClosed  
    Application, 58