Open Accessibility Unreal Plugin

0.3

Generated by Doxygen 1.9.5

1 Hierarchical Index 1

1.1 Class Hierarchy 1

2 Class Index 3

2.1 Class List 3

3 File Index 5

3.1 File List 5

4 Class Documentation 9

4.1 OpenAccessibilityPy.Audio.AudioResampler Class Reference 9

4.1.1 Detailed Description 9

4.1.2 Constructor & Destructor Documentation 9

4.1.2.1 __init__() 9

4.1.2.2 __del__() 10

4.1.3 Member Function Documentation 10

4.1.3.1 resample() 10

4.2 OpenAccessibilityPy.CommunicationServer.CommunicationServer Class Reference 11

4.2.1 Detailed Description 12

4.2.2 Constructor & Destructor Documentation 12

4.2.2.1 __init__() 12

4.2.2.2 __del__() 13

4.2.3 Member Function Documentation 13

4.2.3.1 EventOccured() 13

4.2.3.2 ReceiveJSON() 14

4.2.3.3 ReceiveMultipart() 14

4.2.3.4 ReceiveNDArray() 14

4.2.3.5 ReceiveNDArrayWithMeta() 15

4.2.3.6 ReceiveString() 16

4.2.3.7 RecieveRaw() 16

4.2.3.8 SendJSON() 16

4.2.3.9 SendMultipart() 17

4.2.3.10 SendMultipartWithMeta() 17

4.2.3.11 SendNDArray() 18

4.2.3.12 SendNDArrayWithMeta() 19

4.2.3.13 SendString() 19

4.2.4 Member Data Documentation 20

4.2.4.1 context 20

4.2.4.2 poller 20

4.2.4.3 poller_timeout_time 20

4.2.4.4 recv_socket 20

4.2.4.5 recv_socket_context 20

4.2.4.6 send_socket_context 21

4.3 FAccessibilityNodeFactory Class Reference 21
4.3.1 Detailed Description 21
4.3.2 Constructor & Destructor Documentation 21
4.3.2.1 FAccessibilityNodeFactory() 21
4.3.2.2 ∼FAccessibilityNodeFactory() 22
4.3.3 Member Function Documentation 22
4.3.3.1 CreateNode() 22
4.3.3.2 SetSharedPtr() 23
4.3.3.3 WrapNodeWidget() 23
4.3.3.4 WrapPinWidget() 24
4.4 FAssetAccessibilityRegistry Class Reference 25
4.4.1 Detailed Description 26
4.4.2 Constructor & Destructor Documentation 26
4.4.2.1 FAssetAccessibilityRegistry() 26
4.4.2.2 ∼FAssetAccessibilityRegistry() 26
4.4.3 Member Function Documentation 26
4.4.3.1 GetAllGraphIndexes() [1/2] 27
4.4.3.2 GetAllGraphIndexes() [2/2] 27
4.4.3.3 GetAllGraphKeyIndexes() [1/2] 27
4.4.3.4 GetAllGraphKeyIndexes() [2/2] 28
4.4.3.5 GetGraphIndexer() 28
4.4.3.6 IsGameWorldAssetRegistered() 28
4.4.3.7 IsGraphAssetRegistered() 29
4.4.3.8 RegisterGameWorldAsset() 29
4.4.3.9 RegisterGraphAsset() [1/2] 30
4.4.3.10 RegisterGraphAsset() [2/2] 30
4.4.3.11 UnregisterGameWorldAsset() 31
4.4.3.12 UnregisterGraphAsset() 32
4.4.4 Member Data Documentation 32
4.4.4.1 GraphAssetIndex 33
4.5 FAudioManagerSettings Struct Reference 33
4.5.1 Detailed Description 33
4.5.2 Constructor & Destructor Documentation 33
4.5.2.1 FAudioManagerSettings() 33
4.5.3 Member Data Documentation 33
4.5.3.1 LevelThreshold 34
4.5.3.2 SaveName 34
4.5.3.3 SavePath 34
4.6 FGraphIndexer Class Reference 34
4.6.1 Detailed Description 35
4.6.2 Constructor & Destructor Documentation 35
4.6.2.1 FGraphIndexer() [1/2] 35

4.6.2.2 FGraphIndexer() [2/2] 36
4.6.2.3 ∼FGraphIndexer() 36
4.6.3 Member Function Documentation 36
4.6.3.1 AddNode() [1/2] 36
4.6.3.2 AddNode() [2/2] 37
4.6.3.3 ContainsKey() 37
4.6.3.4 ContainsNode() [1/2] 38
4.6.3.5 ContainsNode() [2/2] 38
4.6.3.6 GetKey() [1/2] 38
4.6.3.7 GetKey() [2/2] 40
4.6.3.8 GetNode() [1/2] 41
4.6.3.9 GetNode() [2/2] 41
4.6.3.10 GetOrAddNode() [1/2] 41
4.6.3.11 GetOrAddNode() [2/2] 42
4.6.3.12 GetPin() [1/2] 42
4.6.3.13 GetPin() [2/2] 43
4.6.3.14 OnGraphEvent() 43
4.6.3.15 OnGraphRebuild() 44
4.6.3.16 RemoveNode() [1/2] 44
4.6.3.17 RemoveNode() [2/2] 45
4.6.4 Member Data Documentation 45
4.6.4.1 AvailableIndices 45
4.6.4.2 IndexMap 46
4.6.4.3 LinkedGraph 46
4.6.4.4 NodeSet 46
4.6.4.5 OnGraphChangedHandle 46
4.7 FGraphLocomotionChunk Struct Reference 46
4.7.1 Detailed Description 47
4.7.2 Member Function Documentation 47
4.7.2.1 GetChunkBottomRight() 47
4.7.2.2 GetChunkBounds() 47
4.7.2.3 GetChunkTopLeft() 47
4.7.2.4 SetChunkBounds() 48
4.7.2.5 SetVisColor() 48
4.7.3 Member Data Documentation 48
4.7.3.1 BottomRight 48
4.7.3.2 ChunkIndexer 48
4.7.3.3 ChunkVisWidget 49
4.7.3.4 ChunkWidget 49
4.7.3.5 TopLeft 49
4.8 FIndexer< KeyType, ValueType > Class Template Reference 49
4.8.1 Detailed Description 50

4.8.2 Constructor & Destructor Documentation 50
4.8.2.1 FIndexer() 50
4.8.2.2 ∼FIndexer() 50
4.8.3 Member Function Documentation 51
4.8.3.1 AddValue() [1/2] 51
4.8.3.2 AddValue() [2/2] 51
4.8.3.3 ContainsKey() 52
4.8.3.4 ContainsValue() 52
4.8.3.5 Empty() 53
4.8.3.6 GetAvailableKey() [1/2] 53
4.8.3.7 GetAvailableKey() [2/2] 53
4.8.3.8 GetKey() [1/2] 54
4.8.3.9 GetKey() [2/2] 54
4.8.3.10 GetKeyOrAddValue() [1/2] 55
4.8.3.11 GetKeyOrAddValue() [2/2] 55
4.8.3.12 GetValue() [1/2] 56
4.8.3.13 GetValue() [2/2] 56
4.8.3.14 IsEmpty() 57
4.8.3.15 Num() [1/2] 57
4.8.3.16 Num() [2/2] 57
4.8.3.17 RemoveValue() [1/2] 58
4.8.3.18 RemoveValue() [2/2] 58
4.8.3.19 Reset() 59
4.8.4 Member Data Documentation 59
4.8.4.1 AvailableIndexes 59
4.8.4.2 IndexMap 59
4.9 FOpenAccessibilityAnalyticsModule Class Reference 60
4.9.1 Detailed Description 60
4.9.2 Member Function Documentation 60
4.9.2.1 DumpTick() 60
4.9.2.2 Get() 61
4.9.2.3 LogEvent() 61
4.9.2.4 ShutdownModule() 62
4.9.2.5 StartupModule() 62
4.9.2.6 SupportsDynamicReloading() 62
4.10 FOpenAccessibilityCommunicationModule Class Reference 62
4.10.1 Detailed Description 63
4.10.2 Member Function Documentation 63
4.10.2.1 Get() 63
4.10.2.2 HandleKeyDownEvent() 64
4.10.2.3 ShutdownModule() 64
4.10.2.4 StartupModule() 64

4.10.2.5 SupportsDynamicReloading() 65
4.10.2.6 Tick() 65
4.10.2.7 TranscribeWaveForm() 65
4.10.3 Member Data Documentation 66
4.10.3.1 AudioManager 66
4.10.3.2 OnTranscriptionRecieved 66
4.10.3.3 PhraseTree 67
4.10.3.4 PhraseTreeUtils 67
4.10.3.5 SocketServer 67
4.11 FOpenAccessibilityModule Class Reference 67
4.11.1 Detailed Description 68
4.11.2 Member Function Documentation 68
4.11.2.1 Get() 68
4.11.2.2 ShutdownModule() 68
4.11.2.3 StartupModule() 69
4.11.2.4 SupportsDynamicReloading() 69
4.11.3 Member Data Documentation 69
4.11.3.1 AccessibilityNodeFactory 69
4.11.3.2 AssetAccessibilityRegistry 70
4.12 FPanelViewPosition Struct Reference 70
4.12.1 Detailed Description 70
4.12.2 Constructor & Destructor Documentation 70
4.12.2.1 FPanelViewPosition() [1/2] 70
4.12.2.2 FPanelViewPosition() [2/2] 71
4.12.3 Member Function Documentation 71
4.12.3.1 operator"!=() [1/2] 71
4.12.3.2 operator"!=() [2/2] 71
4.12.4 Member Data Documentation 71
4.12.4.1 BotRight 71
4.12.4.2 TopLeft 72
4.13 FParseRecord Struct Reference 72
4.13.1 Detailed Description 73
4.13.2 Constructor & Destructor Documentation 73
4.13.2.1 FParseRecord() [1/2] 73
4.13.2.2 FParseRecord() [2/2] 74
4.13.2.3 ∼FParseRecord() 74
4.13.3 Member Function Documentation 74
4.13.3.1 AddPhraseInput() 74
4.13.3.2 AddPhraseString() 74
4.13.3.3 GetContextObj() [1/4] 75
4.13.3.4 GetContextObj() [2/4] 75
4.13.3.5 GetContextObj() [3/4] 75

4.13.3.6 GetContextObj() [4/4] 76
4.13.3.7 GetContextStack() [1/2] 76
4.13.3.8 GetContextStack() [2/2] 77
4.13.3.9 GetPhraseInput() [1/4] 77
4.13.3.10 GetPhraseInput() [2/4] 77
4.13.3.11 GetPhraseInput() [3/4] 78
4.13.3.12 GetPhraseInput() [4/4] 79
4.13.3.13 GetPhraseInputs() [1/2] 79
4.13.3.14 GetPhraseInputs() [2/2] 80
4.13.3.15 GetPhraseString() 80
4.13.3.16 HasContextObj() [1/2] 80
4.13.3.17 HasContextObj() [2/2] 81
4.13.3.18 PopContextObj() [1/2] 81
4.13.3.19 PopContextObj() [2/2] 81
4.13.3.20 PushContextObj() 82
4.13.3.21 RemoveContextObj() 82
4.13.3.22 RemovePhraseInput() 82
4.13.4 Friends And Related Function Documentation 83
4.13.4.1 FPhraseTree 83
4.13.5 Member Data Documentation 83
4.13.5.1 ContextObjectStack 83
4.13.5.2 PhraseInputs 83
4.13.5.3 PhraseRecord 84
4.14 FParseResult Struct Reference 84
4.14.1 Detailed Description 84
4.14.2 Constructor & Destructor Documentation 84
4.14.2.1 FParseResult() [1/3] 84
4.14.2.2 FParseResult() [2/3] 85
4.14.2.3 FParseResult() [3/3] 85
4.14.3 Member Data Documentation 85
4.14.3.1 ReachedNode 85
4.14.3.2 Result 85
4.15 FPhrase2DDirectionalInputNode Class Reference 86
4.15.1 Detailed Description 86
4.15.2 Constructor & Destructor Documentation 86
4.15.2.1 FPhrase2DDirectionalInputNode() [1/5] 86
4.15.2.2 FPhrase2DDirectionalInputNode() [2/5] 87
4.15.2.3 FPhrase2DDirectionalInputNode() [3/5] 87
4.15.2.4 FPhrase2DDirectionalInputNode() [4/5] 87
4.15.2.5 FPhrase2DDirectionalInputNode() [5/5] 87
4.16 FPhraseContextMenuNode< ContextMenuType > Class Template Reference 88
4.16.1 Detailed Description 89

4.16.2 Constructor & Destructor Documentation 89
4.16.2.1 FPhraseContextMenuNode() [1/7] 89
4.16.2.2 FPhraseContextMenuNode() [2/7] 89
4.16.2.3 FPhraseContextMenuNode() [3/7] 89
4.16.2.4 FPhraseContextMenuNode() [4/7] 90
4.16.2.5 FPhraseContextMenuNode() [5/7] 90
4.16.2.6 FPhraseContextMenuNode() [6/7] 90
4.16.2.7 FPhraseContextMenuNode() [7/7] 91
4.16.2.8 ∼FPhraseContextMenuNode() 91
4.16.3 Member Function Documentation 91
4.16.3.1 ConstructContextChildren() 91
4.16.3.2 CreateContextObject() 92
4.16.3.3 HasContextObject() 92
4.16.3.4 ParsePhrase() 93
4.16.3.5 ParsePhraseAsContext() 94
4.16.4 Member Data Documentation 94
4.16.4.1 ContextMenuScalar 95
4.16.4.2 OnGetMenu 95
4.17 FPhraseContextNode< ContextType > Class Template Reference 95
4.17.1 Detailed Description 96
4.17.2 Constructor & Destructor Documentation 96
4.17.2.1 FPhraseContextNode() [1/3] 96
4.17.2.2 FPhraseContextNode() [2/3] 96
4.17.2.3 FPhraseContextNode() [3/3] 97
4.17.2.4 ∼FPhraseContextNode() 97
4.17.3 Member Function Documentation 97
4.17.3.1 ConstructContextChildren() 97
4.17.3.2 CreateContextObject() 98
4.17.3.3 HasContextObject() 98
4.17.3.4 ParsePhrase() 99
4.17.3.5 ParsePhraseAsContext() 99
4.18 FPhraseDirectionalInputNode Class Reference 100
4.18.1 Detailed Description 100
4.18.2 Constructor & Destructor Documentation 101
4.18.2.1 FPhraseDirectionalInputNode() [1/5] 101
4.18.2.2 FPhraseDirectionalInputNode() [2/5] 101
4.18.2.3 FPhraseDirectionalInputNode() [3/5] 101
4.18.2.4 FPhraseDirectionalInputNode() [4/5] 101
4.18.2.5 FPhraseDirectionalInputNode() [5/5] 102
4.19 FPhraseEnumInputNode< TEnum > Class Template Reference 102
4.19.1 Detailed Description 103
4.19.2 Constructor & Destructor Documentation 103

4.19.2.1 FPhraseEnumInputNode() [1/5] . . . 103
4.19.2.2 FPhraseEnumInputNode() [2/5] . . . 103
4.19.2.3 FPhraseEnumInputNode() [3/5] . . . 103
4.19.2.4 FPhraseEnumInputNode() [4/5] . . . 104
4.19.2.5 FPhraseEnumInputNode() [5/5] . . . 104
4.19.2.6 ∼FPhraseEnumInputNode() . . . 104
4.19.3 Member Function Documentation . . . 104
4.19.3.1 MeetsInputRequirements() . . . 104
4.19.3.2 RecordInput() . . . 105
4.20 FPhraseEventNode Class Reference . . . 106
4.20.1 Detailed Description . . . 106
4.20.2 Constructor & Destructor Documentation . . . 106
4.20.2.1 FPhraseEventNode() [1/3] . . . 107
4.20.2.2 FPhraseEventNode() [2/3] . . . 107
4.20.2.3 FPhraseEventNode() [3/3] . . . 107
4.20.2.4 ∼FPhraseEventNode() . . . 107
4.20.3 Member Function Documentation . . . 107
4.20.3.1 IsLeafNode() . . . 108
4.20.3.2 ParsePhrase() . . . 108
4.20.3.3 RequiresPhrase() [1/2] . . . 108
4.20.3.4 RequiresPhrase() [2/2] . . . 109
4.21 FPhraseInputNode< InputType > Class Template Reference . . . 109
4.21.1 Detailed Description . . . 110
4.21.2 Constructor & Destructor Documentation . . . 111
4.21.2.1 FPhraseInputNode() [1/5] . . . 111
4.21.2.2 FPhraseInputNode() [2/5] . . . 111
4.21.2.3 FPhraseInputNode() [3/5] . . . 111
4.21.2.4 FPhraseInputNode() [4/5] . . . 112
4.21.2.5 FPhraseInputNode() [5/5] . . . 112
4.21.2.6 ∼FPhraseInputNode() . . . 112
4.21.3 Member Function Documentation . . . 112
4.21.3.1 MeetsInputRequirements() . . . 112
4.21.3.2 ParsePhrase() . . . 113
4.21.3.3 RecordInput() . . . 114
4.21.3.4 RequiresPhrase() [1/2] . . . 114
4.21.3.5 RequiresPhrase() [2/2] . . . 115
4.21.4 Member Data Documentation . . . 115
4.21.4.1 OnInputReceived . . . 116
4.22 FPhraseNode Class Reference . . . 116
4.22.1 Detailed Description . . . 117
4.22.2 Constructor & Destructor Documentation . . . 117
4.22.2.1 FPhraseNode() [1/4] . . . 117

4.22.2.2 FPhraseNode() [2/4] 118
4.22.2.3 FPhraseNode() [3/4] 118
4.22.2.4 FPhraseNode() [4/4] 118
4.22.2.5 ∼FPhraseNode() 118
4.22.3 Member Function Documentation 119
4.22.3.1 BindChildNode() 119
4.22.3.2 BindChildNodeForce() 119
4.22.3.3 BindChildrenNodes() 120
4.22.3.4 BindChildrenNodesForce() 120
4.22.3.5 CanBindChild() 121
4.22.3.6 HasLeafChild() [1/2] 121
4.22.3.7 HasLeafChild() [2/2] 122
4.22.3.8 IsLeafNode() 122
4.22.3.9 ParseChildren() 122
4.22.3.10 ParsePhrase() 123
4.22.3.11 ParsePhraseAsContext() 124
4.22.3.12 ParsePhraseIfRequired() 124
4.22.3.13 RequiresPhrase() [1/2] 125
4.22.3.14 RequiresPhrase() [2/2] 125
4.22.4 Member Data Documentation 126
4.22.4.1 bHasLeafChild 126
4.22.4.2 BoundPhrase 126
4.22.4.3 ChildNodes 126
4.22.4.4 OnPhraseParsed 126
4.22.4.5 ParentNode 126
4.23 FPhrasePositionalInputNode Class Reference 127
4.23.1 Detailed Description 127
4.23.2 Constructor & Destructor Documentation 127
4.23.2.1 FPhrasePositionalInputNode() [1/5] 127
4.23.2.2 FPhrasePositionalInputNode() [2/5] 128
4.23.2.3 FPhrasePositionalInputNode() [3/5] 128
4.23.2.4 FPhrasePositionalInputNode() [4/5] 128
4.23.2.5 FPhrasePositionalInputNode() [5/5] 128
4.24 FPhraseScrollInputNode Class Reference 129
4.24.1 Detailed Description 129
4.24.2 Constructor & Destructor Documentation 129
4.24.2.1 FPhraseScrollInputNode() [1/5] 129
4.24.2.2 FPhraseScrollInputNode() [2/5] 130
4.24.2.3 FPhraseScrollInputNode() [3/5] 130
4.24.2.4 FPhraseScrollInputNode() [4/5] 130
4.24.2.5 FPhraseScrollInputNode() [5/5] 130
4.25 FPhraseStringInputNode Class Reference 131

4.25.1 Detailed Description . . . 131
4.25.2 Constructor & Destructor Documentation . . . 131
4.25.2.1 FPhraseStringInputNode() [1/4] . . . 132
4.25.2.2 FPhraseStringInputNode() [2/4] . . . 132
4.25.2.3 FPhraseStringInputNode() [3/4] . . . 132
4.25.2.4 FPhraseStringInputNode() [4/4] . . . 132
4.25.2.5 ∼FPhraseStringInputNode() . . . 133
4.25.3 Member Function Documentation . . . 133
4.25.3.1 MeetsInputRequirements() . . . 133
4.25.3.2 RecordInput() . . . 133
4.26 FPhraseTree Class Reference . . . 134
4.26.1 Detailed Description . . . 135
4.26.2 Constructor & Destructor Documentation . . . 135
4.26.2.1 FPhraseTree() . . . 135
4.26.2.2 ∼FPhraseTree() . . . 135
4.26.3 Member Function Documentation . . . 135
4.26.3.1 BindBranch() . . . 135
4.26.3.2 BindBranches() . . . 136
4.26.3.3 GetContextManager() . . . 136
4.26.3.4 ParsePhrase() . . . 136
4.26.3.5 ParseTranscription() . . . 137
4.26.3.6 Tick() . . . 139
4.27 FPhraseTreeBranchBind Struct Reference . . . 139
4.27.1 Detailed Description . . . 140
4.27.2 Constructor & Destructor Documentation . . . 140
4.27.2.1 FPhraseTreeBranchBind() [1/2] . . . 140
4.27.2.2 FPhraseTreeBranchBind() [2/2] . . . 140
4.27.2.3 ∼FPhraseTreeBranchBind() . . . 140
4.27.3 Member Data Documentation . . . 140
4.27.3.1 BranchRoot . . . 141
4.27.3.2 StartNode . . . 141
4.28 FPhraseTreeContextManager Struct Reference . . . 141
4.28.1 Detailed Description . . . 142
4.28.2 Constructor & Destructor Documentation . . . 142
4.28.2.1 FPhraseTreeContextManager() . . . 142
4.28.2.2 ∼FPhraseTreeContextManager() . . . 142
4.28.3 Member Function Documentation . . . 142
4.28.3.1 GetContextStack() . . . 142
4.28.3.2 HasContextObject() . . . 142
4.28.3.3 HasContextObjects() . . . 143
4.28.3.4 IsEmpty() . . . 143
4.28.3.5 PeekContextObject() [1/2] . . . 143

4.28.3.6 PeekContextObject() [2/2] . . . 143
4.28.3.7 PopContextObject() [1/3] . . . 144
4.28.3.8 PopContextObject() [2/3] . . . 144
4.28.3.9 PopContextObject() [3/3] . . . 144
4.28.3.10 PushContextObject() . . . 145
4.28.4 Friends And Related Function Documentation . . . 145
4.28.4.1 FPhraseTree . . . 145
4.29 FSocketCommunicationServer Class Reference . . . 145
4.29.1 Detailed Description . . . 147
4.29.2 Constructor & Destructor Documentation . . . 147
4.29.2.1 FSocketCommunicationServer() . . . 147
4.29.2.2 ∼FSocketCommunicationServer() . . . 148
4.29.3 Member Function Documentation . . . 148
4.29.3.1 DeserializeJSON() . . . 148
4.29.3.2 EventOccured() . . . 148
4.29.3.3 RecvArray() . . . 149
4.29.3.4 RecvJson() . . . 150
4.29.3.5 RecvMultipartWithMeta() . . . 150
4.29.3.6 RecvString() . . . 151
4.29.3.7 RecvStringMultipart() . . . 152
4.29.3.8 RecvStringMultipartWithMeta() . . . 153
4.29.3.9 SendArrayBuffer() [1/3] . . . 153
4.29.3.10 SendArrayBuffer() [2/3] . . . 154
4.29.3.11 SendArrayBuffer() [3/3] . . . 154
4.29.3.12 SendArrayMessage() [1/3] . . . 155
4.29.3.13 SendArrayMessage() [2/3] . . . 156
4.29.3.14 SendArrayMessage() [3/3] . . . 156
4.29.3.15 SendArrayMessageWithMeta() [1/3] . . . 157
4.29.3.16 SendArrayMessageWithMeta() [2/3] . . . 158
4.29.3.17 SendArrayMessageWithMeta() [3/3] . . . 159
4.29.3.18 SendJsonBuffer() . . . 159
4.29.3.19 SendStringBuffer() . . . 160
4.29.3.20 SerializeJSON() . . . 161
4.29.4 Member Data Documentation . . . 161
4.29.4.1 Context . . . 161
4.29.4.2 Poller . . . 161
4.29.4.3 PollTimeout . . . 162
4.29.4.4 RecvAddress . . . 162
4.29.4.5 RecvSocket . . . 162
4.29.4.6 SendAddress . . . 162
4.29.4.7 SendSocket . . . 162
4.30 FTranscriptionVisualizer Class Reference . . . 163

4.30.1 Detailed Description . . . 163
4.30.2 Constructor & Destructor Documentation . . . 163
4.30.2.1 FTranscriptionVisualizer() . . . 164
4.30.2.2 ∼FTranscriptionVisualizer() . . . 164
4.30.3 Member Function Documentation . . . 164
4.30.3.1 ConstructVisualizer() . . . 164
4.30.3.2 GetDisplayVisualizerPosition() . . . 165
4.30.3.3 GetTopScreenVisualizerPosition() . . . 165
4.30.3.4 MoveVisualizer() . . . 165
4.30.3.5 OnTranscriptionRecieved() . . . 166
4.30.3.6 RegisterTicker() . . . 166
4.30.3.7 ReparentWindow() . . . 166
4.30.3.8 Tick() . . . 167
4.30.3.9 UnregisterTicker() . . . 167
4.30.3.10 UpdateVisualizer() . . . 168
4.30.4 Member Data Documentation . . . 168
4.30.4.1 TickDelegateHandle . . . 168
4.30.4.2 VisContent . . . 168
4.30.4.3 VisWindow . . . 168
4.31 UAccessibilityGraphEditorContext::FTreeViewTickRequirements Struct Reference . . . 169
4.31.1 Detailed Description . . . 169
4.31.2 Constructor & Destructor Documentation . . . 169
4.31.2.1 FTreeViewTickRequirements() . . . 169
4.31.3 Member Data Documentation . . . 169
4.31.3.1 PrevNumGeneratedChildren . . . 169
4.31.3.2 PrevNumItemsBeingObserved . . . 169
4.31.3.3 PrevScrollDistance . . . 170
4.31.3.4 PrevSearchText . . . 170
4.32 IPhraseContextNodeBase Class Reference . . . 170
4.32.1 Detailed Description . . . 170
4.32.2 Member Function Documentation . . . 171
4.32.2.1 ConstructContextChildren() . . . 171
4.32.2.2 CreateContextObject() . . . 171
4.32.2.3 HasContextObject() . . . 171
4.33 IPhraseNodeBase Class Reference . . . 172
4.33.1 Detailed Description . . . 172
4.33.2 Member Function Documentation . . . 172
4.33.2.1 HasLeafChild() . . . 172
4.33.2.2 IsLeafNode() . . . 173
4.33.2.3 ParsePhrase() . . . 173
4.33.2.4 ParsePhraseAsContext() . . . 173
4.33.2.5 RequiresPhrase() . . . 174

4.34 OpenAccessibilityPy.Logging.LogLevel Class Reference 174
4.34.1 Detailed Description 174
4.34.2 Member Data Documentation 174
4.34.2.1 ERROR 175
4.34.2.2 INFO 175
4.34.2.3 WARNING 175
4.35 TestWhisper.ModelInfo Class Reference 175
4.35.1 Detailed Description 175
4.36 NumericParser Class Reference 175
4.36.1 Detailed Description 176
4.36.2 Member Function Documentation 176
4.36.2.1 IsValidNumeric() 176
4.36.2.2 StringToNumeric() 176
4.37 OAEditorAccessibilityManager Class Reference 177
4.37.1 Detailed Description 177
4.37.2 Constructor & Destructor Documentation 177
4.37.2.1 OAEditorAccessibilityManager() 177
4.37.2.2 ∼OAEditorAccessibilityManager() 177
4.38 OpenAccessibility Class Reference 177
4.38.1 Detailed Description 178
4.38.2 Constructor & Destructor Documentation 178
4.38.2.1 OpenAccessibility() 178
4.39 OpenAccessibilityAnalytics Class Reference 179
4.39.1 Detailed Description 179
4.39.2 Constructor & Destructor Documentation 179
4.39.2.1 OpenAccessibilityAnalytics() 179
4.40 OpenAccessibilityCommunication Class Reference 180
4.40.1 Detailed Description 180
4.40.2 Constructor & Destructor Documentation 181
4.40.2.1 OpenAccessibilityCommunication() 181
4.41 OpenAccessibilityPy.OpenAccessibilityPy Class Reference 182
4.41.1 Detailed Description 182
4.41.2 Constructor & Destructor Documentation 182
4.41.2.1 __init__() 182
4.41.2.2 __del__() 183
4.41.3 Member Function Documentation 183
4.41.3.1 HandleTranscriptionRequest() 183
4.41.3.2 Shutdown() 184
4.41.3.3 Tick() 185
4.41.4 Member Data Documentation 185
4.41.4.1 audio_resampler 185
4.41.4.2 com_server 186

4.41.4.3 pyshutdown_handle 186
4.41.4.4 tick_handle 186
4.41.4.5 whisper_interface 186
4.41.4.6 worker_pool 186
4.42 SAccessibilityTranscriptionVis Class Reference 186
4.42.1 Detailed Description 187
4.42.2 Constructor & Destructor Documentation 187
4.42.2.1 ∼SAccessibilityTranscriptionVis() 187
4.42.3 Member Function Documentation 187
4.42.3.1 Construct() 188
4.42.3.2 SLATE_BEGIN_ARGS() 188
4.42.3.3 Tick() 189
4.42.3.4 UpdateTopTranscription() 189
4.42.4 Member Data Documentation 189
4.42.4.1 TranscriptionContainer 189
4.42.4.2 TranscriptionSlots 190
4.43 SContentIndexer Class Reference 190
4.43.1 Detailed Description 191
4.43.2 Constructor & Destructor Documentation 191
4.43.2.1 ∼SContentIndexer() 191
4.43.3 Member Function Documentation 191
4.43.3.1 Construct() 191
4.43.3.2 ConstructBottomIndexer() 192
4.43.3.3 ConstructContentContainer() 192
4.43.3.4 ConstructIndexContainer() 193
4.43.3.5 ConstructIndexText() 193
4.43.3.6 ConstructLeftIndexer() 194
4.43.3.7 ConstructRightIndexer() 194
4.43.3.8 ConstructTopIndexer() 195
4.43.3.9 GetContent() [1/2] 196
4.43.3.10 GetContent() [2/2] 196
4.43.3.11 SLATE_BEGIN_ARGS() 196
4.43.3.12 Tick() 197
4.43.3.13 UpdateIndex() 197
4.43.4 Member Data Documentation 197
4.43.4.1 IndexedContent 197
4.43.4.2 IndexerWidget 197
4.44 SIndexer Class Reference 198
4.44.1 Detailed Description 198
4.44.2 Constructor & Destructor Documentation 198
4.44.2.1 ∼SIndexer() 198
4.44.3 Member Function Documentation 199

4.44.3.1 Construct() . . . 199
4.44.3.2 GetIndexText() . . . 199
4.44.3.3 SLATE_BEGIN_ARGS() . . . 199
4.44.3.4 Tick() . . . 200
4.44.3.5 UpdateIndex() [1/3] . . . 200
4.44.3.6 UpdateIndex() [2/3] . . . 200
4.44.3.7 UpdateIndex() [3/3] . . . 201
4.44.4 Member Data Documentation . . . 201
4.44.4.1 IndexTextBlock . . . 201
4.45 TGraphAccessibilityNodeFactory< T > Class Template Reference . . . 201
4.45.1 Detailed Description . . . 202
4.45.2 Constructor & Destructor Documentation . . . 202
4.45.2.1 TGraphAccessibilityNodeFactory() [1/2] . . . 202
4.45.2.2 TGraphAccessibilityNodeFactory() [2/2] . . . 202
4.45.2.3 ∼TGraphAccessibilityNodeFactory() . . . 203
4.45.3 Member Function Documentation . . . 203
4.45.3.1 CreateNodeWidget() . . . 203
4.45.3.2 CreatePinWidget() . . . 204
4.45.4 Member Data Documentation . . . 205
4.45.4.1 AccessibilityRegistry . . . 205
4.45.4.2 Implementation . . . 206
4.46 UAccessibilityAddNodeContextMenu Class Reference . . . 206
4.46.1 Detailed Description . . . 208
4.46.2 Constructor & Destructor Documentation . . . 208
4.46.2.1 UAccessibilityAddNodeContextMenu() [1/4] . . . 208
4.46.2.2 UAccessibilityAddNodeContextMenu() [2/4] . . . 208
4.46.2.3 UAccessibilityAddNodeContextMenu() [3/4] . . . 208
4.46.2.4 UAccessibilityAddNodeContextMenu() [4/4] . . . 209
4.46.2.5 ∼UAccessibilityAddNodeContextMenu() . . . 209
4.46.3 Member Function Documentation . . . 209
4.46.3.1 AppendFilterText() . . . 209
4.46.3.2 AppendScrollDistance() . . . 209
4.46.3.3 ApplyAccessibilityWidget() . . . 210
4.46.3.4 Close() . . . 210
4.46.3.5 DoesItemsRequireRefresh() . . . 211
4.46.3.6 GetFilterText() . . . 211
4.46.3.7 GetGraphActionFromIndex() [1/2] . . . 211
4.46.3.8 GetGraphActionFromIndex() [2/2] . . . 212
4.46.3.9 GetGraphActionFromIndexSP() . . . 212
4.46.3.10 Init() [1/3] . . . 213
4.46.3.11 Init() [2/3] . . . 214
4.46.3.12 Init() [3/3] . . . 214

4.46.3.13 PerformGraphAction() 215
4.46.3.14 RefreshAccessibilityWidgets() 215
4.46.3.15 ResetFilterText() 216
4.46.3.16 ScaleMenu() 216
4.46.3.17 SelectGraphAction() 217
4.46.3.18 SetFilterText() 217
4.46.3.19 SetScrollDistance() 217
4.46.3.20 SetScrollDistanceBottom() 218
4.46.3.21 SetScrollDistanceTop() 218
4.46.3.22 Tick() 218
4.46.3.23 ToggleContextAwareness() 219
4.46.3.24 UpdateAccessibilityWidget() 219
4.46.4 Member Data Documentation 219
4.46.4.1 ContextAwarenessCheckBox 219
4.46.4.2 FilterTextBox 219
4.46.4.3 GraphMenu 220
4.46.4.4 PrevFilterString 220
4.46.4.5 PrevNumGeneratedChildren 220
4.46.4.6 PrevNumItemsBeingObserved 220
4.46.4.7 PrevScrollDistance 220
4.46.4.8 TreeView 220
4.47 UAccessibilityGraphEditorContext Class Reference 221
4.47.1 Detailed Description 222
4.47.2 Constructor & Destructor Documentation 222
4.47.2.1 UAccessibilityGraphEditorContext() 222
4.47.3 Member Function Documentation 222
4.47.3.1 AppendFilterText() 222
4.47.3.2 AppendScrollDistance() 223
4.47.3.3 Close() 223
4.47.3.4 CreateAccessibilityWrapper() 224
4.47.3.5 FindGraphActionMenu() 224
4.47.3.6 FindStaticComponents() 225
4.47.3.7 FindTreeView() 225
4.47.3.8 GetFilterText() 226
4.47.3.9 GetStaticIndexOffset() 226
4.47.3.10 GetTreeViewAction() 226
4.47.3.11 Init() 227
4.47.3.12 ScaleMenu() 228
4.47.3.13 SelectAction() 228
4.47.3.14 SetFilterText() 229
4.47.3.15 SetScrollDistance() 229
4.47.3.16 SetScrollDistanceBottom() 230

4.47.3.17 SetScrollDistanceTop() 230
4.47.3.18 Tick() 230
4.47.3.19 TickTreeViewAccessibility() 231
4.47.3.20 TreeViewCanTick() 231
4.47.3.21 TreeViewRequiresTick() 232
4.47.3.22 UpdateAccessibilityWidget() 232
4.47.4 Member Data Documentation 232
4.47.4.1 CheckBoxes 232
4.47.4.2 FilterTextBox 233
4.47.4.3 GraphMenu 233
4.47.4.4 TreeView 233
4.47.4.5 TreeViewTickRequirements 233
4.48 UAccessibilityGraphLocomotionContext Class Reference 233
4.48.1 Detailed Description 234
4.48.2 Constructor & Destructor Documentation 234
4.48.2.1 UAccessibilityGraphLocomotionContext() 235
4.48.2.2 ∼UAccessibilityGraphLocomotionContext() 235
4.48.3 Member Function Documentation 235
4.48.3.1 BindFocusChangedEvent() 235
4.48.3.2 CalculateVisualChunksBounds() 235
4.48.3.3 CancelLocomotion() 236
4.48.3.4 ChangeChunkVis() 236
4.48.3.5 Close() 236
4.48.3.6 ConfirmSelection() 237
4.48.3.7 CreateVisualGrid() 237
4.48.3.8 GenerateVisualChunks() 237
4.48.3.9 HideNativeVisuals() 238
4.48.3.10 Init() [1/2] 239
4.48.3.11 Init() [2/2] 239
4.48.3.12 MoveViewport() [1/2] 239
4.48.3.13 MoveViewport() [2/2] 240
4.48.3.14 OnFocusChanged() 240
4.48.3.15 RemoveVisualGrid() 240
4.48.3.16 RevertToPreviousView() 241
4.48.3.17 SelectChunk() 241
4.48.3.18 UnbindFocusChangedEvent() 242
4.48.3.19 UnHideNativeVisuals() 242
4.48.4 Member Data Documentation 242
4.48.4.1 ChunkArray 242
4.48.4.2 ChunkSize 242
4.48.4.3 CurrentViewPosition 243
4.48.4.4 GridContainer 243

4.48.4.5 GridParent 243
4.48.4.6 LinkedEditor 243
4.48.4.7 PreviousPositions 243
4.48.4.8 StartViewPosition 243
4.48.4.9 StartViewZoom 244
4.49 UAccessibilityWindowToolbar Class Reference 244
4.49.1 Detailed Description 244
4.49.2 Constructor & Destructor Documentation 244
4.49.2.1 UAccessibilityWindowToolbar() 245
4.49.2.2 ∼UAccessibilityWindowToolbar() 245
4.49.3 Member Function Documentation 245
4.49.3.1 GetActiveToolkitWidget() 245
4.49.3.2 IsActiveToolbar() 245
4.49.3.3 SelectToolbarItem() 246
4.49.3.4 Tick() 247
4.50 UAudioManager Class Reference 247
4.50.1 Detailed Description 248
4.50.2 Constructor & Destructor Documentation 248
4.50.2.1 UAudioManager() 248
4.50.2.2 ∼UAudioManager() 249
4.50.3 Member Function Documentation 249
4.50.3.1 GetAudioCaptureNumChannels() 249
4.50.3.2 GetAudioCaptureSampleRate() 249
4.50.3.3 IsCapturingAudio() 250
4.50.3.4 OnDefaultDeviceChanged() 250
4.50.3.5 PRIVATE_OnAudioGenerate() 250
4.50.3.6 SaveAudioBufferToWAV() 251
4.50.3.7 StartCapturingAudio() 251
4.50.3.8 StopCapturingAudio() 251
4.50.4 Member Data Documentation 252
4.50.4.1 OnAudioReadyForTranscription 252
4.50.4.2 Settings 252
4.51 UBAudioCapture Class Reference 252
4.51.1 Detailed Description 253
4.51.2 Constructor & Destructor Documentation 253
4.51.2.1 UBAudioCapture() 253
4.51.2.2 ∼UBAudioCapture() 253
4.51.3 Member Function Documentation 253
4.51.3.1 OpenDefaultAudioStream() 253
4.52 ULocalizedInputLibrary Class Reference 254
4.52.1 Detailed Description 255
4.52.2 Constructor & Destructor Documentation 255

4.52.2.1 ULocalizedInputLibrary() 255
4.52.2.2 ∼ULocalizedInputLibrary() 255
4.52.3 Member Function Documentation 255
4.52.3.1 BindBranches() 255
4.52.3.2 KeyboardInputAdd() 256
4.52.3.3 KeyboardInputConfirm() 257
4.52.3.4 KeyboardInputExit() 258
4.52.3.5 KeyboardInputRemove() 258
4.52.3.6 KeyboardInputReset() 259
4.53 UNodeInteractionLibrary Class Reference 260
4.53.1 Detailed Description 261
4.53.2 Constructor & Destructor Documentation 261
4.53.2.1 UNodeInteractionLibrary() 261
4.53.2.2 ∼UNodeInteractionLibrary() 261
4.53.3 Member Function Documentation 261
4.53.3.1 BindBranches() 261
4.53.3.2 BlueprintCompile() 265
4.53.3.3 DeleteNode() 266
4.53.3.4 LocomotionCancel() 266
4.53.3.5 LocomotionConfirm() 267
4.53.3.6 LocomotionRevert() 267
4.53.3.7 LocomotionSelect() 268
4.53.3.8 MoveNode() 268
4.53.3.9 NodeAddMenu() 269
4.53.3.10 NodeAddPinMenu() 270
4.53.3.11 NodeAddScroll() 272
4.53.3.12 NodeAddSearchAdd() 272
4.53.3.13 NodeAddSearchRemove() 273
4.53.3.14 NodeAddSearchReset() 273
4.53.3.15 NodeAddSelect() 273
4.53.3.16 NodeIndexFocus() 274
4.53.3.17 PinConnect() 274
4.53.3.18 PinDisconnect() 275
4.53.3.19 SelectionAlignment() 276
4.53.3.20 SelectionComment() 276
4.53.3.21 SelectionMove() 277
4.53.3.22 SelectionNodeToggle() 278
4.53.3.23 SelectionReset() 278
4.53.3.24 SelectionStraighten() 279
4.54 UParseEnumInput Class Reference 279
4.54.1 Detailed Description 280
4.54.2 Constructor & Destructor Documentation 280

4.54.2.1 ∼UParseEnumInput() 280
4.54.3 Member Function Documentation 280
4.54.3.1 GetEnumType() [1/2] 280
4.54.3.2 GetEnumType() [2/2] 280
4.54.3.3 SetEnumType() 281
4.54.4 Member Data Documentation 281
4.54.4.1 EnumType 281
4.55 UParseInput Class Reference 281
4.55.1 Detailed Description 282
4.55.2 Constructor & Destructor Documentation 282
4.55.2.1 ∼UParseInput() 282
4.56 UParseIntInput Class Reference 282
4.56.1 Detailed Description 283
4.56.2 Constructor & Destructor Documentation 283
4.56.2.1 ∼UParseIntInput() 283
4.56.3 Member Function Documentation 283
4.56.3.1 GetValue() [1/2] 283
4.56.3.2 GetValue() [2/2] 283
4.56.3.3 SetValue() 284
4.56.4 Member Data Documentation 284
4.56.4.1 Value 284
4.57 UParseStringInput Class Reference 284
4.57.1 Detailed Description 285
4.57.2 Constructor & Destructor Documentation 285
4.57.2.1 ∼UParseStringInput() 285
4.57.3 Member Function Documentation 285
4.57.3.1 GetValue() [1/2] 285
4.57.3.2 GetValue() [2/2] 285
4.57.3.3 SetValue() 286
4.57.4 Member Data Documentation 286
4.57.4.1 Value 286
4.58 UPhraseTreeContextMenuObject Class Reference 286
4.58.1 Detailed Description 287
4.58.2 Constructor & Destructor Documentation 287
4.58.2.1 UPhraseTreeContextMenuObject() [1/2] 288
4.58.2.2 UPhraseTreeContextMenuObject() [2/2] 288
4.58.2.3 ∼UPhraseTreeContextMenuObject() 288
4.58.3 Member Function Documentation 288
4.58.3.1 BindMenuDismissed() 288
4.58.3.2 BindTickDelegate() 289
4.58.3.3 Close() 289
4.58.3.4 GetWindow() 289

4.58.3.5 Init() [1/2] 289
4.58.3.6 Init() [2/2] 290
4.58.3.7 OnMenuDismissed() 290
4.58.3.8 RemoveMenuDismissed() 291
4.58.3.9 RemoveTickDelegate() 291
4.58.3.10 ScaleMenu() 291
4.58.3.11 SetMenu() 292
4.58.3.12 Tick() 292
4.58.4 Member Data Documentation 292
4.58.4.1 Menu 292
4.58.4.2 Window 293
4.59 UPhraseTreeContextObject Class Reference 293
4.59.1 Detailed Description 293
4.59.2 Constructor & Destructor Documentation 294
4.59.2.1 UPhraseTreeContextObject() 294
4.59.2.2 ∼UPhraseTreeContextObject() 294
4.59.3 Member Function Documentation 294
4.59.3.1 Close() 294
4.59.3.2 GetContextRoot() 294
4.59.3.3 GetIsActive() 295
4.59.3.4 SetContextRootNode() 295
4.59.4 Member Data Documentation 295
4.59.4.1 bIsActive 295
4.59.4.2 ContextRoot 296
4.60 UPhraseTreeFunctionLibrary Class Reference 296
4.60.1 Detailed Description 296
4.60.2 Member Function Documentation 296
4.60.2.1 BindBranches() 296
4.61 UPhraseTreeUtils Class Reference 297
4.61.1 Detailed Description 297
4.61.2 Constructor & Destructor Documentation 297
4.61.2.1 UPhraseTreeUtils() 297
4.61.2.2 ∼UPhraseTreeUtils() 297
4.61.3 Member Function Documentation 298
4.61.3.1 RegisterFunctionLibrary() 298
4.61.3.2 SetPhraseTree() 299
4.61.4 Member Data Documentation 299
4.61.4.1 PhraseTree 299
4.61.4.2 RegisteredLibraries 300
4.62 UViewInteractionLibrary Class Reference 300
4.62.1 Detailed Description 300
4.62.2 Constructor & Destructor Documentation 300

4.62.2.1 UViewInteractionLibrary() 300
4.62.2.2 ∼UViewInteractionLibrary() 301
4.62.3 Member Function Documentation 301
4.62.3.1 BindBranches() 301
4.62.3.2 IndexFocus() 302
4.62.3.3 MoveViewport() 302
4.62.3.4 ZoomViewport() 303
4.63 UWindowInteractionLibrary Class Reference 304
4.63.1 Detailed Description 305
4.63.2 Constructor & Destructor Documentation 305
4.63.2.1 UWindowInteractionLibrary() 305
4.63.2.2 ∼UWindowInteractionLibrary() 305
4.63.3 Member Function Documentation 305
4.63.3.1 BindBranches() 305
4.63.3.2 CloseActiveWindow() 306
4.63.3.3 SelectToolBarItem() 307
4.63.4 Member Data Documentation 307
4.63.4.1 WindowToolBar 307
4.64 OpenAccessibilityPy.WhisperInterface.WhisperInterface Class Reference 307
4.64.1 Detailed Description 308
4.64.2 Constructor & Destructor Documentation 308
4.64.2.1 __init__() 308
4.64.2.2 __del__() 309
4.64.3 Member Function Documentation 309
4.64.3.1 process_audio_buffer() 309
4.64.3.2 process_file_from_dir() 310
4.64.4 Member Data Documentation 310
4.64.4.1 beam_size 310
4.64.4.2 whisper_model 310

5 File Documentation **311**
5.1 init_unreal.py 311
5.2 old_init_unreal.py 312
5.3 __init__.py 313
5.4 __main__.py 315
5.5 Audio.py 316
5.6 CommunicationServer.py 318
5.7 LibUtils.py 322
5.8 Logging.py 322
5.9 WhisperInterface.py 323
5.10 TestWhisper.py 324
5.11 OpenAccessibility.Build.cs 325

5.12 SAccessibilityTranscriptionVis.cpp 326
5.13 SContentIndexer.cpp 327
5.14 SIndexer.cpp 329
5.15 AccessibilityAddNodeContextMenu.cpp 330
5.16 AccessibilityGraphEditorContext.cpp 334
5.17 AccessibilityGraphLocomotionContext.cpp 339
5.18 AccessibilityWindowToolbar.cpp 343
5.19 AssetAccessibilityRegistry.cpp 347
5.20 GraphIndexer.cpp 350
5.21 OAccessibilityNodeFactory.cpp 353
5.22 OAEditorAccessibilityManager.cpp 356
5.23 OpenAccessibility.cpp 356
5.24 LocalizedInputLibrary.cpp 360
5.25 NodeInteractionLibrary.cpp 363
5.26 ViewInteractionLibrary.cpp 375
5.27 WindowInteractionLibrary.cpp 377
5.28 TranscriptionVisualizer.cpp 378
5.29 WidgetUtils.h 380
5.30 AccessibilityNodeFactory.h 382
5.31 SAccessibilityTranscriptionVis.h 384
5.32 SContentIndexer.h 384
5.33 SIndexer.h 385
5.34 AccessibilityAddNodeContextMenu.h 386
5.35 AccessibilityGraphEditorContext.h 387
5.36 AccessibilityGraphLocomotionContext.h 389
5.37 AccessibilityWindowToolbar.h 391
5.38 AssetAccessibilityRegistry.h 392
5.39 GraphIndexer.h 393
5.40 Indexer.h 394
5.41 OAccessibilityNodeFactory.h 396
5.42 OAEditorAccessibilityManager.h 397
5.43 OpenAccessibility.h 397
5.44 OpenAccessibilityLogging.h 398
5.45 LocalizedInputLibrary.h 398
5.46 NodeInteractionLibrary.h 399
5.47 ViewInteractionLibrary.h 400
5.48 WindowInteractionLibrary.h 400
5.49 TranscriptionVisualizer.h 401
5.50 OpenAccessibilityAnalytics.Build.cs 402
5.51 OpenAccessibilityAnalytics.cpp 402
5.52 OpenAccessibilityAnalyticsLogging.h 404
5.53 OpenAccessibilityAnalytics.h 404

5.54 OpenAccessibilityCommunication.Build.cs 406
5.55 AudioManager.cpp 407
5.56 OpenAccessibilityComLogging.cpp 408
5.57 OpenAccessibilityCommunication.cpp 408
5.58 PhraseTree.cpp 411
5.59 ContextMenuObject.cpp 414
5.60 PhraseEnumInputNode.cpp 415
5.61 PhraseEventNode.cpp 416
5.62 PhraseInputNode.cpp 417
5.63 PhraseNode.cpp 418
5.64 PhraseStringInputNode.cpp 421
5.65 Utils.cpp 422
5.66 PhraseTreeUtils.cpp 423
5.67 SocketCommunicationServer.cpp 423
5.68 UBAudioCapture.cpp 429
5.69 AudioManager.h 430
5.70 OpenAccessibilityComLogging.h 431
5.71 OpenAccessibilityCommunication.h 431
5.72 PhraseTree.h 432
5.73 ContextMenuObject.h 434
5.74 ContextObject.h 435
5.75 InputContainers.h 436
5.76 UParseEnumInput.h 437
5.77 UParseInput.h 437
5.78 UParseIntInput.h 438
5.79 UParseStringInput.h 438
5.80 ParseRecord.h 439
5.81 ParseResult.h 442
5.82 IPhraseContextNode.h 442
5.83 PhraseContextMenuNode.h 442
5.84 PhraseContextNode.h 445
5.85 PhraseDirectionalInputNode.h 447
5.86 PhraseEnumInputNode.h 449
5.87 PhraseEventNode.h 449
5.88 PhraseInputNode.h 450
5.89 PhraseNode.h 450
5.90 PhraseStringInputNode.h 451
5.91 PhraseTreeFunctionLibrary.h 452
5.92 Utils.h 452
5.93 Utils.h 454
5.94 PhraseTreeUtils.h 454
5.95 SocketCommunicationServer.h 455

5.96 UBAudioCapture.h 456

Index **457**

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

OpenAccessibilityPy.Audio.AudioResampler 9
OpenAccessibilityPy.CommunicationServer.CommunicationServer 11
FAssetAccessibilityRegistry 25
FAudioManagerSettings 33
FGraphIndexer 34
FGraphLocomotionChunk 46
FGraphNodeFactory
    TGraphAccessibilityNodeFactory< T > 201
FGraphPanelNodeFactory
    FAccessibilityNodeFactory 21
FIndexer< KeyType, ValueType > 49
FIndexer< int32, SMultiBlockBaseWidget ∗ > 49
FPanelViewPosition 70
FParseRecord 72
FParseResult 84
FPhraseTreeBranchBind 139
FPhraseTreeContextManager 141
FSocketCommunicationServer 145
FTranscriptionVisualizer 163
UAccessibilityGraphEditorContext::FTreeViewTickRequirements 169
IModuleInterface
    FOpenAccessibilityAnalyticsModule 60
    FOpenAccessibilityCommunicationModule 62
    FOpenAccessibilityModule 67
IPhraseContextNodeBase 170
    FPhraseContextMenuNode< ContextMenuType > 88
    FPhraseContextNode< ContextType > 95
IPhraseNodeBase 172
TestWhisper.ModelInfo 175
ModuleRules
    OpenAccessibility 177
    OpenAccessibilityAnalytics 179
    OpenAccessibilityCommunication 180
NumericParser 175
OAEditorAccessibilityManager 177

OpenAccessibilityPy.OpenAccessibilityPy . . . 182
SBox
  SAccessibilityTranscriptionVis . . . 186
  SContentIndexer . . . 190
  SIndexer . . . 198
TSharedFromThis
  FPhraseNode . . . 116
    FPhraseInputNode< int32 > . . . 109
      FPhraseEnumInputNode< EPhrase2DDirectionalInput > . . . 102
        FPhrase2DDirectionalInputNode . . . 86
      FPhraseEnumInputNode< EPhraseDirectionalInput > . . . 102
        FPhraseDirectionalInputNode . . . 100
      FPhraseEnumInputNode< EPhrasePositionalInput > . . . 102
        FPhrasePositionalInputNode . . . 127
      FPhraseEnumInputNode< EPhraseScrollInput > . . . 102
        FPhraseScrollInputNode . . . 129
      FPhraseEnumInputNode< TEnum > . . . 102
    FPhraseInputNode< FString > . . . 109
      FPhraseStringInputNode . . . 131
    FPhraseContextMenuNode< ContextMenuType > . . . 88
    FPhraseContextNode< ContextType > . . . 95
    FPhraseEventNode . . . 106
    FPhraseInputNode< InputType > . . . 109
    FPhraseTree . . . 134
UAudioCapture
  UBAudioCapture . . . 252
UObject
  UAccessibilityWindowToolbar . . . 244
  UAudioManager . . . 247
  UParseInput . . . 281
    UParseIntInput . . . 282
      UParseEnumInput . . . 279
    UParseStringInput . . . 284
  UPhraseTreeContextObject . . . 293
    UAccessibilityGraphLocomotionContext . . . 233
    UPhraseTreeContextMenuObject . . . 286
      UAccessibilityAddNodeContextMenu . . . 206
      UAccessibilityGraphEditorContext . . . 221
  UPhraseTreeFunctionLibrary . . . 296
    ULocalizedInputLibrary . . . 254
    UNodeInteractionLibrary . . . 260
    UViewInteractionLibrary . . . 300
    UWindowInteractionLibrary . . . 304
  UPhraseTreeUtils . . . 297
OpenAccessibilityPy.WhisperInterface.WhisperInterface . . . 307
Enum
  OpenAccessibilityPy.Logging.LogLevel . . . 174

Chapter 2

Class Index

2.1 Class List

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

OpenAccessibilityPy.Audio.AudioResampler 9
OpenAccessibilityPy.CommunicationServer.CommunicationServer 11
FAccessibilityNodeFactory 21
FAssetAccessibilityRegistry 25
FAudioManagerSettings 33
FGraphIndexer 34
FGraphLocomotionChunk 46
FIndexer< KeyType, ValueType > 49
FOpenAccessibilityAnalyticsModule 60
FOpenAccessibilityCommunicationModule 62
FOpenAccessibilityModule 67
FPanelViewPosition 70
FParseRecord
The Collected Information from the Propogation of the Phrase through the tree. 72
FParseResult
Contains the Result of Propagation through the Phrase Tree. 84
FPhrase2DDirectionalInputNode 86
FPhraseContextMenuNode< ContextMenuType > 88
FPhraseContextNode< ContextType > 95
FPhraseDirectionalInputNode 100
FPhraseEnumInputNode< TEnum > 102
FPhraseEventNode 106
FPhraseInputNode< InputType > 109
FPhraseNode 116
FPhrasePositionalInputNode 127
FPhraseScrollInputNode 129
FPhraseStringInputNode 131
FPhraseTree 134
FPhraseTreeBranchBind 139
FPhraseTreeContextManager 141
FSocketCommunicationServer 145
FTranscriptionVisualizer 163
UAccessibilityGraphEditorContext::FTreeViewTickRequirements 169
IPhraseContextNodeBase
Base Abstract Class For Phrase Context Nodes, that are required to have a Context Node. . . 170

IPhraseNodeBase 172
OpenAccessibilityPy.Logging.LogLevel 174
TestWhisper.ModelInfo 175
NumericParser 175
OAEditorAccessibilityManager 177
OpenAccessibility 177
OpenAccessibilityAnalytics 179
OpenAccessibilityCommunication 180
OpenAccessibilityPy.OpenAccessibilityPy 182
SAccessibilityTranscriptionVis 186
SContentIndexer 190
SIndexer 198
TGraphAccessibilityNodeFactory< T > 201
UAccessibilityAddNodeContextMenu 206
UAccessibilityGraphEditorContext 221
UAccessibilityGraphLocomotionContext 233
UAccessibilityWindowToolbar 244
UAudioManager 247
UBAudioCapture 252
ULocalizedInputLibrary 254
UNodeInteractionLibrary 260
UParseEnumInput 279
UParseInput 281
UParseIntInput 282
UParseStringInput 284
UPhraseTreeContextMenuObject 286
UPhraseTreeContextObject 293
UPhraseTreeFunctionLibrary 296
UPhraseTreeUtils 297
UViewInteractionLibrary 300
UWindowInteractionLibrary 304
OpenAccessibilityPy.WhisperInterface.WhisperInterface 307

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Content/Python/init_unreal.py 311
Content/Python/old_init_unreal.py 312
Content/Python/TestWhisper.py 324
Content/Python/OpenAccessibilityPy/__init__.py 313
Content/Python/OpenAccessibilityPy/__main__.py 315
Content/Python/OpenAccessibilityPy/Audio.py 316
Content/Python/OpenAccessibilityPy/CommunicationServer.py 318
Content/Python/OpenAccessibilityPy/LibUtils.py 322
Content/Python/OpenAccessibilityPy/Logging.py 322
Content/Python/OpenAccessibilityPy/WhisperInterface.py 323
Source/OpenAccessibility/OpenAccessibility.Build.cs 325
Source/OpenAccessibility/Private/AssetAccessibilityRegistry.cpp 347
Source/OpenAccessibility/Private/GraphIndexer.cpp 350
Source/OpenAccessibility/Private/OAccessibilityNodeFactory.cpp 353
Source/OpenAccessibility/Private/OAEditorAccessibilityManager.cpp 356
Source/OpenAccessibility/Private/OpenAccessibility.cpp 356
Source/OpenAccessibility/Private/TranscriptionVisualizer.cpp 378
Source/OpenAccessibility/Private/AccessibilityWidgets/SAccessibilityTranscriptionVis.cpp 326
Source/OpenAccessibility/Private/AccessibilityWidgets/SContentIndexer.cpp 327
Source/OpenAccessibility/Private/AccessibilityWidgets/SIndexer.cpp 329
Source/OpenAccessibility/Private/AccessibilityWrappers/AccessibilityAddNodeContextMenu.cpp 330
Source/OpenAccessibility/Private/AccessibilityWrappers/AccessibilityGraphEditorContext.cpp 334
Source/OpenAccessibility/Private/AccessibilityWrappers/AccessibilityGraphLocomotionContext.cpp . . . 339
Source/OpenAccessibility/Private/AccessibilityWrappers/AccessibilityWindowToolbar.cpp 343
Source/OpenAccessibility/Private/PhraseEvents/LocalizedInputLibrary.cpp 360
Source/OpenAccessibility/Private/PhraseEvents/NodeInteractionLibrary.cpp 363
Source/OpenAccessibility/Private/PhraseEvents/ViewInteractionLibrary.cpp 375
Source/OpenAccessibility/Private/PhraseEvents/WindowInteractionLibrary.cpp 377
Source/OpenAccessibility/Private/Utils/WidgetUtils.h 380
Source/OpenAccessibility/Public/AccessibilityNodeFactory.h 382
Source/OpenAccessibility/Public/AssetAccessibilityRegistry.h 392
Source/OpenAccessibility/Public/GraphIndexer.h 393
Source/OpenAccessibility/Public/OAccessibilityNodeFactory.h 396
Source/OpenAccessibility/Public/OAEditorAccessibilityManager.h 397
Source/OpenAccessibility/Public/OpenAccessibility.h 397

Source/OpenAccessibility/Public/OpenAccessibilityLogging.h . . . 398
Source/OpenAccessibility/Public/TranscriptionVisualizer.h . . . 401
Source/OpenAccessibility/Public/AccessibilityWidgets/SAccessibilityTranscriptionVis.h . . . 384
Source/OpenAccessibility/Public/AccessibilityWidgets/SContentIndexer.h . . . 384
Source/OpenAccessibility/Public/AccessibilityWidgets/SIndexer.h . . . 385
Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityAddNodeContextMenu.h . . . 386
Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityGraphEditorContext.h . . . 387
Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityGraphLocomotionContext.h . . . 389
Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityWindowToolbar.h . . . 391
Source/OpenAccessibility/Public/Indexers/Indexer.h . . . 394
Source/OpenAccessibility/Public/PhraseEvents/LocalizedInputLibrary.h . . . 398
Source/OpenAccessibility/Public/PhraseEvents/NodeInteractionLibrary.h . . . 399
Source/OpenAccessibility/Public/PhraseEvents/Utils.h . . . 452
Source/OpenAccessibility/Public/PhraseEvents/ViewInteractionLibrary.h . . . 400
Source/OpenAccessibility/Public/PhraseEvents/WindowInteractionLibrary.h . . . 400
Source/OpenAccessibilityAnalytics/OpenAccessibilityAnalytics.Build.cs . . . 402
Source/OpenAccessibilityAnalytics/Private/OpenAccessibilityAnalytics.cpp . . . 402
Source/OpenAccessibilityAnalytics/Private/OpenAccessibilityAnalyticsLogging.h . . . 404
Source/OpenAccessibilityAnalytics/Public/OpenAccessibilityAnalytics.h . . . 404
Source/OpenAccessibilityCommunication/OpenAccessibilityCommunication.Build.cs . . . 406
Source/OpenAccessibilityCommunication/Private/AudioManager.cpp . . . 407
Source/OpenAccessibilityCommunication/Private/OpenAccessibilityComLogging.cpp . . . 408
Source/OpenAccessibilityCommunication/Private/OpenAccessibilityCommunication.cpp . . . 408
Source/OpenAccessibilityCommunication/Private/PhraseTree.cpp . . . 411
Source/OpenAccessibilityCommunication/Private/PhraseTreeUtils.cpp . . . 423
Source/OpenAccessibilityCommunication/Private/SocketCommunicationServer.cpp . . . 423
Source/OpenAccessibilityCommunication/Private/UBAudioCapture.cpp . . . 429
Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseEnumInputNode.cpp . . . 415
Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseEventNode.cpp . . . 416
Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseInputNode.cpp . . . 417
Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseNode.cpp . . . 418
Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseStringInputNode.cpp . . . 421
Source/OpenAccessibilityCommunication/Private/PhraseTree/Utils.cpp . . . 422
Source/OpenAccessibilityCommunication/Private/PhraseTree/Containers/ContextMenuObject.cpp . . . 414
Source/OpenAccessibilityCommunication/Public/AudioManager.h . . . 430
Source/OpenAccessibilityCommunication/Public/OpenAccessibilityComLogging.h . . . 431
Source/OpenAccessibilityCommunication/Public/OpenAccessibilityCommunication.h . . . 431
Source/OpenAccessibilityCommunication/Public/PhraseTree.h . . . 432
Source/OpenAccessibilityCommunication/Public/PhraseTreeUtils.h . . . 454
Source/OpenAccessibilityCommunication/Public/SocketCommunicationServer.h . . . 455
Source/OpenAccessibilityCommunication/Public/UBAudioCapture.h . . . 456
Source/OpenAccessibilityCommunication/Public/PhraseTree/IPhraseContextNode.h . . . 442
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseContextMenuNode.h . . . 442
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseContextNode.h . . . 445
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseDirectionalInputNode.h . . . 447
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseEnumInputNode.h . . . 449
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseEventNode.h . . . 449
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseInputNode.h . . . 450
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseNode.h . . . 450
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseStringInputNode.h . . . 451
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseTreeFunctionLibrary.h . . . 452
Source/OpenAccessibilityCommunication/Public/PhraseTree/Utils.h . . . 454
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/ContextMenuObject.h . . . 434
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/ContextObject.h . . . 435
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/ParseRecord.h . . . 439
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/ParseResult.h . . . 442
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/Input/InputContainers.h . . . 436
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/Input/UParseEnumInput.h . . . 437

Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/Input/UParseInput.h 437
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/Input/UParseIntInput.h 438
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/Input/UParseStringInput.h . . . 438

Chapter 4

Class Documentation

4.1 OpenAccessibilityPy.Audio.AudioResampler Class Reference

Public Member Functions

- def __init__ (self, int target_sample_rate=16000)
- def __del__ (self)
- np.ndarray resample (self, np.ndarray audio_data, int buffer_sample_rate=48000, int buffer_num_↩ channels=2)

4.1.1 Detailed Description

```
Audio Resampler for Resampling Incoming Audio to the Target Sample Rate. Using FFmpeg for Resampling.
```

Definition at line 15 of file Audio.py.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 __init__()

```
def OpenAccessibilityPy.Audio.AudioResampler.__init__ (
              self,
             int  target_sample_rate = 16000 )
```

```
Constructor of Audio Resampler Class

Args:
    target_sample_rate (int, optional): The Target for all incoming resampling requests. Defaults to 16000 (Re
```

Definition at line 18 of file Audio.py.

```
00018     def __init__(self, target_sample_rate: int = 16000):
00019         """Constructor of Audio Resampler Class
00020 
00021         Args:
00022             target_sample_rate (int, optional): The Target for all incoming resampling requests.
      Defaults to 16000 (Required by Whisper).
00023         """
00024 
00025         self._audio_resampler = av.AudioResampler(
00026             format="s16", layout="mono", rate=target_sample_rate
00027         )
00028         self._resample_mutex = Lock()
00029 
```

4.1.2.2 __del__()

```
def OpenAccessibilityPy.Audio.AudioResampler.__del__ (
                self )
```

```
Destructor of Audio Resampler Class.

Ensures PyAV Resampler Object is Properly Deleted, calling Garbage Collection in the process.
```

Definition at line 30 of file Audio.py.

```
00030     def __del__(self):
00031         """Destructor of Audio Resampler Class.
00032 
00033         Ensures PyAV Resampler Object is Properly Deleted, calling Garbage Collection in the process.
00034         """
00035 
00036         # Try Deleting the resampler object to cleanly free up memory
00037         try:
00038             del self._audio_resampler
00039         except:
00040             pass
00041 
00042         try:  # Delete the mutex
00043             del self._resample_mutex
00044         except:
00045             pass
00046 
00047         # Force Garbage Collection, due to resampler not being properly deleted otherwise.
00048         gc.collect()
00049 
```

4.1.3 Member Function Documentation

4.1.3.1 resample()

```
np.ndarray OpenAccessibilityPy.Audio.AudioResampler.resample (
                self,
               np.ndarray audio_data,
               int  buffer_sample_rate = 48000,
               int  buffer_num_channels = 2 )
```

```
Resamples the Incoming Audio Data to the Classes Assigned Target Sample Rate.

Args:
    audio_data (np.ndarray): Audio Data to Resample.
    buffer_sample_rate (int, optional): Sample Rate of the Incoming Audio Data. Defaults to 48000.
    buffer_num_channels (int, optional): Number of Channels in the Incoming Audio Data. Defaults to 2 (Stereo)

Returns:
    np.ndarray: Resampled Version of the Incoming Audio Data.
```

Definition at line 50 of file Audio.py.

```
00055     ) -> np.ndarray:
00056         """Resamples the Incoming Audio Data to the Classes Assigned Target Sample Rate.
00057 
00058         Args:
00059             audio_data (np.ndarray): Audio Data to Resample.
00060             buffer_sample_rate (int, optional): Sample Rate of the Incoming Audio Data. Defaults to
       48000.
00061             buffer_num_channels (int, optional): Number of Channels in the Incoming Audio Data.
       Defaults to 2 (Stereo).
00062 
```

```
        Returns:
            np.ndarray: Resampled Version of the Incoming Audio Data.
        """

        audio_data = self._convert_to_s16(audio_data).reshape(-1, 1)

        frame: av.AudioFrame = av.AudioFrame.from_ndarray(
            audio_data.T,
            format="s16",
            layout="stereo" if buffer_num_channels == 2 else "mono",
        )

        frame.sample_rate = buffer_sample_rate

        resampled_frames: list[av.AudioFrame] = []
        with self._resample_mutex:
            resampled_frames = self._audio_resampler.resample(frame)

        return self._convert_to_float32(resampled_frames[0].to_ndarray()).reshape(
            -1,
        )

```

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

- Content/Python/OpenAccessibilityPy/Audio.py

4.2 OpenAccessibilityPy.CommunicationServer.CommunicationServer Class Reference

Public Member Functions

- def __init__ (self, int send_socket_type, int recv_socket_type, str send_socket_addr="tcp://127.0.0.1:5556", str recv_socket_addr="tcp://127.0.0.1:5555", int poll_timeout=10)
- def __del__ (self)
- bool EventOccured (self)
- bool SendString (self, str message)
- bool SendJSON (self, dict message)
- bool SendNDArray (self, np.ndarray message)
- bool SendNDArrayWithMeta (self, np.ndarray message, dict meta)
- bool SendMultipart (self, list message)
- bool SendMultipartWithMeta (self, list message, dict meta)
- def RecieveRaw (self)
- str ReceiveString (self)
- def ReceiveJSON (self)
- np.ndarray ReceiveNDArray (self, dtype=np.float32)
- tuple[np.ndarray, dict] ReceiveNDArrayWithMeta (self, dtype=np.float32)
- list[bytes] ReceiveMultipart (self)

Public Attributes

- context
- send_socket_context
- recv_socket
- recv_socket_context
- poller
- poller_timeout_time

4.2.1 Detailed Description

```
Communication Server Class for Handling Communication Between Python and C++.

Using ZeroMQ for Socket Communication. (Push / PULL Architecture)
```

Definition at line 11 of file CommunicationServer.py.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 __init__()

```
def OpenAccessibilityPy.CommunicationServer.CommunicationServer.__init__ (
                self,
                int send_socket_type,
                int recv_socket_type,
                str  send_socket_addr = "tcp://127.0.0.1:5556",
                str  recv_socket_addr = "tcp://127.0.0.1:5555",
                int  poll_timeout = 10 )
```

```
Constructor of Communication Server Class

Args:
    send_socket_type (int): ZeroMQ Socket Type for Sending Messages.
    recv_socket_type (int): ZeroMQ Socket Type for Receiving Messages.
    send_socket_addr (str, optional): Local Address / Port for Sending Communication Data. Defaults to "tcp://
    recv_socket_addr (str, optional): Local Address / Port for Receiving Communication Data. Defaults to "tcp:
    poll_timeout (int, optional): Amount of time (ms) for event polling on the Receive Socket. Defaults to 10.
```

Definition at line 17 of file CommunicationServer.py.

```
    ):
        """Constructor of Communication Server Class

        Args:
            send_socket_type (int): ZeroMQ Socket Type for Sending Messages.
            recv_socket_type (int): ZeroMQ Socket Type for Receiving Messages.
            send_socket_addr (str, optional): Local Address / Port for Sending Communication Data.
    Defaults to "tcp://127.0.0.1:5556".
            recv_socket_addr (str, optional): Local Address / Port for Receiving Communication Data.
    Defaults to "tcp://127.0.0.1:5555".
            poll_timeout (int, optional): Amount of time (ms) for event polling on the Receive Socket.
    Defaults to 10.
        """

        # Create the Context
        self.context = zmq.Context()

        # Create a Socket
        self.send_socket: zmq.Socket = self.context.socket(send_socket_type)
        self.send_socket_context = self.send_socket.connect(send_socket_addr)

        self.recv_socket = self.context.socket(recv_socket_type)
        self.recv_socket_context = self.recv_socket.bind(recv_socket_addr)

        self.poller = zmq.Poller()
        self.poller.register(self.recv_socket, zmq.POLLIN)
        self.poller_timeout_time = poll_timeout

```

4.2.2.2 __del__()

```
def OpenAccessibilityPy.CommunicationServer.CommunicationServer.__del__ (
              self )
```

```
Destructor of Communication Server Class.

Closes the Sockets and Terminates the ZeroMQ Context.
```

Definition at line 49 of file CommunicationServer.py.

```
    def __del__(self):
        """Destructor of Communication Server Class.

        Closes the Sockets and Terminates the ZeroMQ Context.
        """

        self.send_socket.close()
        self.recv_socket.close()

        self.context.term()

```

4.2.3 Member Function Documentation

4.2.3.1 EventOccured()

```
bool OpenAccessibilityPy.CommunicationServer.CommunicationServer.EventOccured (
              self )
```

```
Checks if a Receive Event has Occured on the Receive Socket.

Returns:
    bool: True if an Event has Occured, False Otherwise.
```

Definition at line 60 of file CommunicationServer.py.

```
    def EventOccured(self) -> bool:
        """Checks if a Receive Event has Occured on the Receive Socket.

        Returns:
            bool: True if an Event has Occured, False Otherwise.
        """

        polled_events = dict(self.poller.poll(self.poller_timeout_time))
        if len(polled_events) > 0 and polled_events.get(self.recv_socket) == zmq.POLLIN:
            return True
        else:
            return False

```

4.2.3.2 ReceiveJSON()

```
def OpenAccessibilityPy.CommunicationServer.CommunicationServer.ReceiveJSON (
              self )
```

```
Receive a JSON Message from the Receive Socket.

Returns:
    dict: Dictionary of the Received JSON Message.
```

Definition at line 211 of file CommunicationServer.py.

```
    def ReceiveJSON(self):
        """Receive a JSON Message from the Receive Socket.

        Returns:
            dict: Dictionary of the Received JSON Message.
        """

        return json.loads(self.recv_socket.recv_json(zmq.DONTWAIT))

```

4.2.3.3 ReceiveMultipart()

```
list[bytes] OpenAccessibilityPy.CommunicationServer.CommunicationServer.ReceiveMultipart (
              self )
```

```
Receieved a Raw Multipart Message from the Receive Socket.

Returns:
    list[bytes]: Raw List of Bytes from the Received Multipart Message.
```

Definition at line 262 of file CommunicationServer.py.

```
    def ReceiveMultipart(self) -> list[bytes]:
        """Receieved a Raw Multipart Message from the Receive Socket.

        Returns:
            list[bytes]: Raw List of Bytes from the Received Multipart Message.
        """

        return self.recv_socket.recv_multipart(zmq.DONTWAIT)
```

4.2.3.4 ReceiveNDArray()

```
np.ndarray OpenAccessibilityPy.CommunicationServer.CommunicationServer.ReceiveNDArray (
              self,
              dtype = np.float32 )
```

```
Receives a Numpy NDArray from the Receive Socket.

Args:
    dtype (optional): Type of NDArray of Received Data. Defaults to np.float32.

Returns:
    np.ndarray: Receieved NDArray Message.
```

Definition at line 220 of file CommunicationServer.py.

```
    def ReceiveNDArray(self, dtype=np.float32) -> np.ndarray:
        """Receives a Numpy NDArray from the Receive Socket.

        Args:
            dtype (optional): Type of NDArray of Received Data. Defaults to np.float32.

        Returns:
            np.ndarray: Receieved NDArray Message.
        """

        return np.frombuffer(
            self.recv_socket.recv(zmq.DONTWAIT),
            dtype=dtype,
        )

```

4.2.3.5 ReceiveNDArrayWithMeta()

tuple[np.ndarray, dict] OpenAccessibilityPy.CommunicationServer.CommunicationServer.Receive↩NDArrayWithMeta (
self,
dtype = np.float32)

```
Receives a Numpy NDArray with Metadata from the Receive Socket.

Args:
    dtype (optional): Type of NDArray of Received Data. Defaults to np.float32.

Returns:
    tuple[np.ndarray, dict]: Tuple of Received NDArray and Dict Metadata Object.
```

Definition at line 235 of file CommunicationServer.py.

```
    def ReceiveNDArrayWithMeta(self, dtype=np.float32) -> tuple[np.ndarray, dict]:
        """Receives a Numpy NDArray with Metadata from the Receive Socket.

        Args:
            dtype (optional): Type of NDArray of Received Data. Defaults to np.float32.

        Returns:
            tuple[np.ndarray, dict]: Tuple of Received NDArray and Dict Metadata Object.
        """

        recv_message = self.recv_socket.recv_multipart(zmq.DONTWAIT)

        if len(recv_message) > 1:
            return (
                np.frombuffer(recv_message[1], dtype=dtype),
                json.loads(recv_message[0]),
            )

        elif len(recv_message) == 1:
            Log(
                "CommunicationServer | Error Receiving NDArray With Meta. Only Contains One Message,
      Assumed Data.",
                LogLevel.WARNING,
            )
            return (np.frombuffer(recv_message[0], dtype=dtype), {})

        Log("CommunicationServer | Error Receiving NDArray With Meta", LogLevel.WARNING)

```

4.2.3.6 ReceiveString()

```
str OpenAccessibilityPy.CommunicationServer.CommunicationServer.ReceiveString (
              self )
```

```
Receives a String Message from the Receive Socket.

Returns:
    str: Received String Message.
```

Definition at line 202 of file CommunicationServer.py.

```
    def ReceiveString(self) -> str:
        """Receives a String Message from the Receive Socket.

        Returns:
            str: Received String Message.
        """

        return self.recv_socket.recv_string(zmq.DONTWAIT)

```

4.2.3.7 RecieveRaw()

```
def OpenAccessibilityPy.CommunicationServer.CommunicationServer.RecieveRaw (
              self )
```

```
Receives a Raw Message of Bytes from the Receive Socket.

Returns:
    bytes: Raw Received Bytes from the Receive Socket.
```

Definition at line 193 of file CommunicationServer.py.

```
    def RecieveRaw(self):
        """Receives a Raw Message of Bytes from the Receive Socket.

        Returns:
            bytes: Raw Received Bytes from the Receive Socket.
        """

        return self.recv_socket.recv(zmq.DONTWAIT)

```

4.2.3.8 SendJSON()

```
bool OpenAccessibilityPy.CommunicationServer.CommunicationServer.SendJSON (
              self,
             dict message )
```

```
Sends a JSON Message on the Send Socket.

Args:
    message (dict): Stringified JSON Message to Send.

Returns:
    bool: True if the Message was Sent Successfully, False Otherwise.
```

Definition at line 90 of file CommunicationServer.py.

```
    def SendJSON(self, message: dict) -> bool:
        """Sends a JSON Message on the Send Socket.

        Args:
            message (dict): Stringified JSON Message to Send.

        Returns:
            bool: True if the Message was Sent Successfully, False Otherwise.
        """

        try:
            self.send_socket.send_json(message)
            return True
        except:
            Log(
                "CommunicationServer | Error Sending JSON Message",
                LogLevel.WARNING,
            )
            return False

```

4.2.3.9 SendMultipart()

```
bool OpenAccessibilityPy.CommunicationServer.CommunicationServer.SendMultipart (
                self,
                list message )
```

```
Sends a Multipart Message on the Send Socket.

Args:
    message (list): List of Messages to Send.

Returns:
    bool: True if the MultiPart Message was Sent Successfully, False Otherwise.
```

Definition at line 152 of file CommunicationServer.py.

```
    def SendMultipart(self, message: list) -> bool:
        """Sends a Multipart Message on the Send Socket.

        Args:
            message (list): List of Messages to Send.

        Returns:
            bool: True if the MultiPart Message was Sent Successfully, False Otherwise.
        """

        try:
            self.send_socket.send_multipart(message)
            return True
        except:
            Log(
                "CommunicationServer | Error Sending Multipart Message",
                LogLevel.WARNING,
            )
            return False

```

4.2.3.10 SendMultipartWithMeta()

```
bool OpenAccessibilityPy.CommunicationServer.CommunicationServer.SendMultipartWithMeta (
                self,
                list message,
                dict meta )
```

```
Sends a Multipart Message with Metadata on the Send Socket.

Args:
    message (list): List of Messages to Send.
    meta (dict): Metadata to Send.

Returns:
    bool: True if the MultiPart Message with Metadata was Sent Successfully, False Otherwise.
```

Definition at line 172 of file CommunicationServer.py.

```
    def SendMultipartWithMeta(self, message: list, meta: dict) -> bool:
        """Sends a Multipart Message with Metadata on the Send Socket.

        Args:
            message (list): List of Messages to Send.
            meta (dict): Metadata to Send.

        Returns:
            bool: True if the MultiPart Message with Metadata was Sent Successfully, False Otherwise.
        """

        try:
            self.send_socket.send_multipart([json.dumps(meta).encode(), *message])
            return True
        except:
            Log(
                "CommunicationServer | Error Sending Multipart With Meta Message",
                LogLevel.WARNING,
            )
            return False

```

4.2.3.11 SendNDArray()

bool OpenAccessibilityPy.CommunicationServer.CommunicationServer.SendNDArray (
self,
np.ndarray *message*)

```
Sends a Numpy NDArray Message on the Send Socket.

Args:
    message (np.ndarray): NDArray of Data to Send.

Returns:
    bool: True if the Data was Sent Successfully, False Otherwise.
```

Definition at line 110 of file CommunicationServer.py.

```
    def SendNDArray(self, message: np.ndarray) -> bool:
        """Sends a Numpy NDArray Message on the Send Socket.

        Args:
            message (np.ndarray): NDArray of Data to Send.

        Returns:
            bool: True if the Data was Sent Successfully, False Otherwise.
        """

        try:
            self.send_socket.send(message)
            return True
        except:
            Log(
                "CommunicationServer | Error Sending NDArray Message",
                LogLevel.WARNING,
            )
            return False

```

4.2.3.12 SendNDArrayWithMeta()

```
bool OpenAccessibilityPy.CommunicationServer.CommunicationServer.SendNDArrayWithMeta (
              self,
             np.ndarray message,
             dict meta )
```

```
Sends a Numpy NDArray Message with Metadata on the Send Socket.

Args:
    message (np.ndarray): NDArray of Data to Send.
    meta (dict): A Dictionary of Metadata to Send.

Returns:
    bool: True if the Data was Sent Successfully, False Otherwise.
```

Definition at line 130 of file CommunicationServer.py.

```
    def SendNDArrayWithMeta(self, message: np.ndarray, meta: dict) -> bool:
        """Sends a Numpy NDArray Message with Metadata on the Send Socket.

        Args:
            message (np.ndarray): NDArray of Data to Send.
            meta (dict): A Dictionary of Metadata to Send.

        Returns:
            bool: True if the Data was Sent Successfully, False Otherwise.
        """

        try:
            self.send_socket.send_multipart([json.dumps(meta).encode(), message.data])

            return True
        except:
            Log(
                "CommunicationServer | Error Sending NDArray With Meta Message",
                LogLevel.WARNING,
            )
            return False

```

4.2.3.13 SendString()

```
bool OpenAccessibilityPy.CommunicationServer.CommunicationServer.SendString (
              self,
             str message )
```

```
Sends a String Message on the Send Socket.

Args:
    message (str): String Message to Send.

Returns:
    bool: True if the Message was Sent Successfully, False Otherwise.
```

Definition at line 73 of file CommunicationServer.py.

```
    def SendString(self, message: str) -> bool:
        """Sends a String Message on the Send Socket.

        Args:
            message (str): String Message to Send.

        Returns:
            bool: True if the Message was Sent Successfully, False Otherwise.
        """

        try:
            self.send_socket.send_string(message)
            return True
        except:
            Log("CommunicationServer | Error Sending String Message", LogLevel.WARNING)
            return False

```

4.2.4 Member Data Documentation

4.2.4.1 context

`OpenAccessibilityPy.CommunicationServer.CommunicationServer.context`

Definition at line 36 of file CommunicationServer.py.

4.2.4.2 poller

`OpenAccessibilityPy.CommunicationServer.CommunicationServer.poller`

Definition at line 45 of file CommunicationServer.py.

4.2.4.3 poller_timeout_time

`OpenAccessibilityPy.CommunicationServer.CommunicationServer.poller_timeout_time`

Definition at line 47 of file CommunicationServer.py.

4.2.4.4 recv_socket

`OpenAccessibilityPy.CommunicationServer.CommunicationServer.recv_socket`

Definition at line 42 of file CommunicationServer.py.

4.2.4.5 recv_socket_context

`OpenAccessibilityPy.CommunicationServer.CommunicationServer.recv_socket_context`

Definition at line 43 of file CommunicationServer.py.

4.2.4.6 send_socket_context

```
OpenAccessibilityPy.CommunicationServer.CommunicationServer.send_socket_context
```

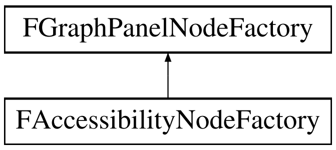
Definition at line 40 of file CommunicationServer.py.

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

- Content/Python/OpenAccessibilityPy/CommunicationServer.py

4.3 FAccessibilityNodeFactory Class Reference

Inheritance diagram for FAccessibilityNodeFactory:



Public Member Functions

- virtual TSharedPtr< class SGraphNode > CreateNode (UEdGraphNode ∗Node) const override
- void WrapNodeWidget (UEdGraphNode ∗Node, TSharedRef< SGraphNode > NodeWidget, int NodeIndex) const

    Wraps the Node Widget with Accessibility Indexing.
- void WrapPinWidget (UEdGraphPin ∗Pin, TSharedRef< SGraphPin > PinWidget, int PinIndex, SGraphNode ∗OwnerNode) const

    Wraps the Pin Widget with Accessibility Indexing.
- void SetSharedPtr (TSharedPtr< FAccessibilityNodeFactory > InSharedPtr)

4.3.1 Detailed Description

Definition at line 11 of file OAccessibilityNodeFactory.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 FAccessibilityNodeFactory()

```
FAccessibilityNodeFactory::FAccessibilityNodeFactory ( )
```

Definition at line 23 of file OAccessibilityNodeFactory.cpp.

```
00023                                                       : FGraphPanelNodeFactory()
00024 {
00025     UE_LOGFMT(LogOpenAccessibility, Display, "Accessibility Node Factory Constructed");
00026 }
```

4.3.2.2 ∼FAccessibilityNodeFactory()

```
FAccessibilityNodeFactory::∼FAccessibilityNodeFactory ( )
```

Definition at line 28 of file OAccessibilityNodeFactory.cpp.

```
{

}
```

4.3.3 Member Function Documentation

4.3.3.1 CreateNode()

```
TSharedPtr< class SGraphNode > FAccessibilityNodeFactory::CreateNode (
            UEdGraphNode * Node ) const  [override], [virtual]
```

Definition at line 33 of file OAccessibilityNodeFactory.cpp.

```
{
    UE_LOG(LogOpenAccessibility, Display, TEXT("Accessibility Node Factory Used to construct %s"),
    *InNode->GetName());

    check(InNode);

    // Hack to get around the possible infinite loop of using
    // this factory to create the node from the factory itself.
    FEdGraphUtilities::UnregisterVisualNodeFactory(FOpenAccessibilityModule::Get().AccessibilityNodeFactory);
    TSharedPtr<SGraphNode> OutNode = FNodeFactory::CreateNodeWidget(InNode);
    FEdGraphUtilities::RegisterVisualNodeFactory(FOpenAccessibilityModule::Get().AccessibilityNodeFactory);

    // Get Node Accessibility Index, from registry
    TSharedRef<FGraphIndexer> GraphIndexer = FOpenAccessibilityModule::Get()
        .AssetAccessibilityRegistry->GetGraphIndexer(InNode->GetGraph());

    int NodeIndex = -1;
    GraphIndexer->GetOrAddNode(InNode, NodeIndex);

    {
        // Create Accessibility Widgets For Pins
        TArray<UEdGraphPin*> Pins = InNode->GetAllPins();
        TSharedPtr<SGraphPin> PinWidget;

        for (int i = 0; i < Pins.Num(); i++)
        {
            UEdGraphPin* Pin = Pins[i];

            PinWidget = OutNode->FindWidgetForPin(Pin);
            if (!PinWidget.IsValid())
            {
                continue;
            }

            WrapPinWidget(Pin, PinWidget.ToSharedRef(), i, OutNode.Get());
        }

        PinWidget.Reset();
    }

    // Wrap The Node Widget
    WrapNodeWidget(InNode, OutNode.ToSharedRef(), NodeIndex);

    return OutNode;
}
```

4.3.3.2 SetSharedPtr()

```
void FAccessibilityNodeFactory::SetSharedPtr (
              TSharedPtr< FAccessibilityNodeFactory > InSharedPtr )  [inline]
```

Definition at line 40 of file OAccessibilityNodeFactory.h.

```
    {
        ThisPtr = InSharedPtr;
    }
```

4.3.3.3 WrapNodeWidget()

```
void FAccessibilityNodeFactory::WrapNodeWidget (
              UEdGraphNode * Node,
              TSharedRef< SGraphNode > NodeWidget,
              int NodeIndex ) const  [inline]
```

Wraps the Node Widget with Accessibility Indexing.

Parameters

Node	The Node Object That is Being Wrapped.
NodeWidget	The Node Widget That is Being Wrapped.
NodeIndex	The Index of the Node.

Definition at line 79 of file OAccessibilityNodeFactory.cpp.

```
{
    TSharedRef<SWidget> WidgetToWrap = NodeWidget->GetSlot(ENodeZone::Center)->GetWidget();
    check(WidgetToWrap != SNullWidget::NullWidget);

    NodeWidget->GetOrAddSlot(ENodeZone::Center)
        .HAlign(HAlign_Fill)
        [
            SNew(SVerticalBox)

                + SVerticalBox::Slot()
                .HAlign(HAlign_Fill)
                .AutoHeight()
                .Padding(FMargin(1.5f, 0.25f))
                [
                    SNew(SOverlay)

                        + SOverlay::Slot()
                        [
                            SNew(SImage)
                                .Image(FAppStyle::Get().GetBrush("Graph.Node.Body"))
                        ]

                        + SOverlay::Slot()
                        .Padding(FMargin(4.0f, 0.0f))
                        [
                            SNew(SHorizontalBox)
                                + SHorizontalBox::Slot()
                                .HAlign(HAlign_Right)
                                .VAlign(VAlign_Center)
                                .Padding(1.f)
                                [
                                    SNew(SOverlay)
                                        + SOverlay::Slot()
                                        [
                                            SNew(SIndexer)
                                            .IndexValue(NodeIndex)
                                            .TextColor(FLinearColor::White)
                                            .BorderColor(FLinearColor::Gray)
                                        ]
                                ]
                        ]
                ]

                + SVerticalBox::Slot()
                .HAlign(HAlign_Fill)
                .AutoHeight()
                [
                    WidgetToWrap
                ]
        ];
}
```

4.3.3.4 WrapPinWidget()

```
void FAccessibilityNodeFactory::WrapPinWidget (
            UEdGraphPin * Pin,
            TSharedRef< SGraphPin > PinWidget,
            int PinIndex,
            SGraphNode * OwnerNode ) const [inline]
```

Wraps the Pin Widget with Accessibility Indexing.

Parameters

Pin	The Pin Object That is Being Wrapped.
PinWidget	The Node Widget That is Being Wrapped.
PinIndex	The Index of the Pin.
OwnerNode	The Owning Node Widget of the Pin.

Definition at line 132 of file OAccessibilityNodeFactory.cpp.

```
{
    TSharedRef<SWidget> PinWidgetContent = PinWidget->GetContent();
    check(PinWidgetContent != SNullWidget::NullWidget);

    TSharedRef<SWidget> AccessibilityWidget = SNew(SOverlay)
        .Visibility_Lambda([OwnerNode]() -> EVisibility {

            if (OwnerNode->HasAnyUserFocusOrFocusedDescendants() || OwnerNode->IsHovered() || OwnerNode->GetOwnerPanel()->SelectionManager.IsNodeSelected(OwnerNode->GetNodeObj()))
                return EVisibility::Visible;

            return EVisibility::Hidden;
        })
        + SOverlay::Slot()
        [
            SNew(SIndexer)
            .IndexValue(PinIndex)
            .TextColor(FLinearColor::White)
            .BorderColor(FLinearColor::Gray)
        ];

    switch (Pin->Direction)
    {
        case EEdGraphPinDirection::EGPD_Input:
        {
            PinWidget->SetContent(
                SNew(SHorizontalBox)
                    + SHorizontalBox::Slot()
                    .AutoWidth()
                    [
                        PinWidgetContent
                    ]
                    + SHorizontalBox::Slot()
                    .AutoWidth()
                    [
                        AccessibilityWidget
                    ]
            );

            break;
        }

        case EEdGraphPinDirection::EGPD_Output:
        {
            PinWidget->SetContent(
                SNew(SHorizontalBox)
                    + SHorizontalBox::Slot()
                    .AutoWidth()
                    [
                        AccessibilityWidget
                    ]
                    + SHorizontalBox::Slot()
                    .AutoWidth()
                    [
                        PinWidgetContent
                    ]
            );
            break;
        }

        default:
        {
            UE_LOG(LogOpenAccessibility, Error, TEXT("Pin Direction Not Recognized"));
            break;
        }
    }
}
```

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

- Source/OpenAccessibility/Public/OAccessibilityNodeFactory.h
- Source/OpenAccessibility/Private/OAccessibilityNodeFactory.cpp

4.4 FAssetAccessibilityRegistry Class Reference

Public Member Functions

- bool IsGraphAssetRegistered (const UEdGraph ∗InGraph) const

  Checks if the provied graph asset has been registered with the registry.
- bool RegisterGraphAsset (const UEdGraph ∗InGraph)

  Registers the provided graph asset with the registry.
- bool RegisterGraphAsset (const UEdGraph ∗InGraph, const TSharedRef< FGraphIndexer > InGraph←Indexer)
- bool UnregisterGraphAsset (const UEdGraph ∗InGraph)

  Unregisters the provided graph asset from the registry.
- TSharedRef< FGraphIndexer > GetGraphIndexer (const UEdGraph ∗InGraph) const

  Gets the Graph Indexer linked to the provided graph asset.
- void GetAllGraphKeyIndexes (TArray< FGuid > &OutGraphKeys) const

  Gets the Guids of all the Graphs that have been registered with the registry.
- TArray< FGuid > GetAllGraphKeyIndexes () const

  Gets the Guids of all the Graphs that have been registered with the registry.
- void GetAllGraphIndexes (TArray< TSharedPtr< FGraphIndexer > > &OutGraphIndexes) const

  Gets all the Graph Indexers that have been registered with the registry.
- TArray< TSharedPtr< FGraphIndexer > > GetAllGraphIndexes ()

  Gets all the Graph Indexers that have been registered with the registry.
- bool IsGameWorldAssetRegistered (const UWorld ∗InWorld) const

  Checks if the provided world asset has been registered with the registry.
- bool RegisterGameWorldAsset (const UWorld ∗InWorld)

  Registered the UWorld Asset with the Registry.
- bool UnregisterGameWorldAsset (const UWorld ∗InWorld)

  Unregisters the provided UWorld Asset from the Registry.

Public Attributes

- TMap< FGuid, TSharedPtr< FGraphIndexer > > GraphAssetIndex

  A Map containing all the Graph Indexers that have been created for the registered Graph Assets.

4.4.1 Detailed Description

Definition at line 14 of file AssetAccessibilityRegistry.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 FAssetAccessibilityRegistry()

```
FAssetAccessibilityRegistry::FAssetAccessibilityRegistry ( )
```

Definition at line 15 of file AssetAccessibilityRegistry.cpp.

```
{
    GraphAssetIndex = TMap<FGuid, TSharedPtr<FGraphIndexer»();
    //GameWorldAssetIndex = TMap<FGuid, FGameWorldIndexer*>();

    AssetOpenedInEditorHandle =
  GEditor->GetEditorSubsystem<UAssetEditorSubsystem>()->OnAssetOpenedInEditor()
        .AddRaw(this, &FAssetAccessibilityRegistry::OnAssetOpenedInEditor);

    AssetEditorRequestCloseHandle =
  GEditor->GetEditorSubsystem<UAssetEditorSubsystem>()->OnAssetEditorRequestClose()
        .AddRaw(this, &FAssetAccessibilityRegistry::OnAssetEditorRequestClose);
}
```

4.4.2.2 ∼FAssetAccessibilityRegistry()

```
FAssetAccessibilityRegistry::∼FAssetAccessibilityRegistry ( )
```

Definition at line 27 of file AssetAccessibilityRegistry.cpp.

```
{
    GEditor->GetEditorSubsystem<UAssetEditorSubsystem>()->OnAssetOpenedInEditor()
        .Remove(AssetOpenedInEditorHandle);

    GEditor->GetEditorSubsystem<UAssetEditorSubsystem>()->OnAssetEditorRequestClose()
        .Remove(AssetEditorRequestCloseHandle);

    EmptyGraphAssetIndex();
}
```

4.4.3 Member Function Documentation

4.4.3.1 GetAllGraphIndexes() `[1/2]`

```
TArray< TSharedPtr< FGraphIndexer > > FAssetAccessibilityRegistry::GetAllGraphIndexes ( )
```

Gets all the Graph Indexers that have been registered with the registry.

Returns

An Array of all the Found Graph Indexers registered with the registry.

Definition at line 150 of file AssetAccessibilityRegistry.cpp.

```
{
    TArray<TSharedPtr<FGraphIndexer» GraphIndexArray;

    GraphAssetIndex.GenerateValueArray(GraphIndexArray);

    return GraphIndexArray;
}
```

4.4.3.2 GetAllGraphIndexes() `[2/2]`

```
void FAssetAccessibilityRegistry::GetAllGraphIndexes (
            TArray< TSharedPtr< FGraphIndexer > > & OutGraphIndexes ) const
```

Gets all the Graph Indexers that have been registered with the registry.

Parameters

OutGraphIndexes	The Array to apply all the registered graph indexers.

Definition at line 145 of file AssetAccessibilityRegistry.cpp.

```
{
    return GraphAssetIndex.GenerateValueArray(OutGraphIndexes);
}
```

4.4.3.3 GetAllGraphKeyIndexes() `[1/2]`

```
TArray< FGuid > FAssetAccessibilityRegistry::GetAllGraphKeyIndexes ( ) const
```

Gets the Guids of all the Graphs that have been registered with the registry.

Returns

An Array of all Found Guids registered with the registry.

Definition at line 137 of file AssetAccessibilityRegistry.cpp.

```
{
    TArray<FGuid> GraphKeys;
    GraphAssetIndex.GetKeys(GraphKeys);

    return GraphKeys;
}
```

4.4.3.4 GetAllGraphKeyIndexes() [2/2]

```
void FAssetAccessibilityRegistry::GetAllGraphKeyIndexes (
            TArray< FGuid > & OutGraphKeys ) const
```

Gets the Guids of all the Graphs that have been registered with the registry.

Parameters

OutGraphKeys	The Array of Guids to Apply the found Guids to.

Definition at line 132 of file AssetAccessibilityRegistry.cpp.

```
{
    GraphAssetIndex.GetKeys(OutGraphKeys);
}
```

4.4.3.5 GetGraphIndexer()

```
TSharedRef< FGraphIndexer > FAssetAccessibilityRegistry::GetGraphIndexer (
            const UEdGraph * InGraph ) const [inline]
```

Gets the Graph Indexer linked to the provided graph asset.

Parameters

InGraph	The Graph to Find the Linked Indexer For.

Returns

Returns the Found SharedReference of the GraphIndexer when found successfully. Returns nullptr on fail.

Definition at line 50 of file AssetAccessibilityRegistry.h.

```
                                                                        {
        if (GraphAssetIndex.Contains(InGraph->GraphGuid))
            return GraphAssetIndex[InGraph->GraphGuid].ToSharedRef();

        return TSharedRef<FGraphIndexer>();
    }
```

4.4.3.6 IsGameWorldAssetRegistered()

```
bool FAssetAccessibilityRegistry::IsGameWorldAssetRegistered (
            const UWorld * InWorld ) const
```

Checks if the provided world asset has been registered with the registry.

Parameters

InWorld	The UWorld Asset to Check if Registered

Returns

True, if the UWorld Asset is Registered. False, if the UWorld Asset is not.

Definition at line 159 of file AssetAccessibilityRegistry.cpp.

```
{
    throw std::exception("The method or operation is not implemented.");
}
```

4.4.3.7 IsGraphAssetRegistered()

```
bool FAssetAccessibilityRegistry::IsGraphAssetRegistered (
            const UEdGraph * InGraph ) const
```

Checks if the provied graph asset has been registered with the registry.

Parameters

InGraph	The Graph Asset to Check.

Returns

True, if the graph has been registered. False, if the graph has not.

Definition at line 71 of file AssetAccessibilityRegistry.cpp.

```
{
    return GraphAssetIndex.Contains(InUEdGraph->GraphGuid);
}
```

4.4.3.8 RegisterGameWorldAsset()

```
bool FAssetAccessibilityRegistry::RegisterGameWorldAsset (
            const UWorld * InWorld )
```

Registered the UWorld Asset with the Registry.

Parameters

InWorld	The UWorld Asset to Register.

Returns

True, if the Asset was Successfully Registered. False, if the asset could not be registered.

Definition at line 164 of file AssetAccessibilityRegistry.cpp.

```
{
    throw std::exception("The method or operation is not implemented.");
}
```

4.4.3.9 RegisterGraphAsset() [1/2]

```
bool FAssetAccessibilityRegistry::RegisterGraphAsset (
            const UEdGraph * InGraph )
```

Registers the provided graph asset with the registry.

Parameters

InGraph	The Graph to Register.

Returns

True, if the Graph was Successfully Registered. False, if the Graph Could Not Be Registered.

Definition at line 76 of file AssetAccessibilityRegistry.cpp.

```
{
    if (!InGraph->IsValidLowLevel())
        return false;

    GraphAssetIndex.Add(InGraph->GraphGuid, MakeShared<FGraphIndexer>(InGraph));

    for (auto& ChildGraph : InGraph->SubGraphs)
    {
        if (!RegisterGraphAsset(ChildGraph))
        {
            UE_LOG(LogOpenAccessibility, Error, TEXT("|| AssetRegistry || Error When Logging Child
      Graph: { %s } From Parent: { %s }||"), *ChildGraph->GetName(), *InGraph->GetName())

            return false;
        }
    }

    return true;
}
```

4.4.3.10 RegisterGraphAsset() [2/2]

```
bool FAssetAccessibilityRegistry::RegisterGraphAsset (
            const UEdGraph * InGraph,
            const TSharedRef< FGraphIndexer > InGraphIndexer )
```

Definition at line 96 of file AssetAccessibilityRegistry.cpp.

```
{
    if (!InGraph->IsValidLowLevel())
        return false;

    GraphAssetIndex.Add(InGraph->GraphGuid, InGraphIndexer.ToSharedPtr());

    for (auto& ChildGraph : InGraph->SubGraphs)
    {
        if (!RegisterGraphAsset(ChildGraph))
        {
            UE_LOG(LogOpenAccessibility, Error, TEXT("|| AssetRegistry || Error When Logging Child
      Graph: { %s } From Parent: { %s} ||"), *ChildGraph->GetName(), *InGraph->GetName());
            return false;
        }
    }

    return true;
}
```

4.4.3.11 UnregisterGameWorldAsset()

```
bool FAssetAccessibilityRegistry::UnregisterGameWorldAsset (
             const UWorld * InWorld )
```

Unregisters the provided UWorld Asset from the Registry.

Parameters

InWorld	The UWorld Asset to Unregister.

Returns

True, if the Asset was Successfully Registered. False, if the asset could not be registered.

Definition at line 169 of file AssetAccessibilityRegistry.cpp.

```
{
    throw std::exception("The method or operation is not implemented.");
}
```

4.4.3.12 UnregisterGraphAsset()

```
bool FAssetAccessibilityRegistry::UnregisterGraphAsset (
            const UEdGraph * InGraph )
```

Unregisters the provided graph asset from the registry.

Parameters

InGraph	The Graph To Unregister.

Returns

True, if the provided graph was unregistered successfully. False, if the Graph Could Not Be Unregistered.

Definition at line 115 of file AssetAccessibilityRegistry.cpp.

```
{
    GraphAssetIndex.Remove(UEdGraph->GraphGuid);

    for (auto& ChildGraph : UEdGraph->SubGraphs)
    {
        if (!UnregisterGraphAsset(ChildGraph))
        {
            UE_LOG(LogOpenAccessibility, Error, TEXT("|| AssetRegistry || Error When Unregistering
       Child Graph: { %s } From Parent: { %s }||"), *ChildGraph->GetName(), *UEdGraph->GetName())

            return false;
        }
    }

    return true;
}
```

4.4.4 Member Data Documentation

4.4.4.1 GraphAssetIndex

`TMap<FGuid, TSharedPtr<FGraphIndexer> > FAssetAccessibilityRegistry::GraphAssetIndex`

A Map containing all the Graph Indexers that have been created for the registered Graph Assets.

Definition at line 162 of file AssetAccessibilityRegistry.h.

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

- Source/OpenAccessibility/Public/AssetAccessibilityRegistry.h
- Source/OpenAccessibility/Private/AssetAccessibilityRegistry.cpp

4.5 FAudioManagerSettings Struct Reference

Public Attributes

- float LevelThreshold
- FString SaveName

    The Name of the Audio File to be saved to.
- FString SavePath

    The Path to save recorded audio files to.

4.5.1 Detailed Description

Definition at line 15 of file AudioManager.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 FAudioManagerSettings()

`FAudioManagerSettings::FAudioManagerSettings ( ) [inline]`

Definition at line 20 of file AudioManager.h.

```
    {
        // Default Settings
        LevelThreshold = -2.5f;
        SaveName = FString("Captured_User_Audio");
        SavePath = FString("./OpenAccessibility/Audioclips/");
    }
```

4.5.3 Member Data Documentation

4.5.3.1 LevelThreshold

```
float FAudioManagerSettings::LevelThreshold
```

Definition at line 30 of file AudioManager.h.

4.5.3.2 SaveName

```
FString FAudioManagerSettings::SaveName
```

The Name of the Audio File to be saved to.

Definition at line 36 of file AudioManager.h.

4.5.3.3 SavePath

```
FString FAudioManagerSettings::SavePath
```

The Path to save recorded audio files to.

Definition at line 42 of file AudioManager.h.

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

- Source/OpenAccessibilityCommunication/Public/AudioManager.h

4.6 FGraphIndexer Class Reference

Public Member Functions

- FGraphIndexer (const UEdGraph ∗GraphToIndex)
- bool ContainsKey (const int &InKey)

  Checks if the Provided Key is Contained within the Indexer.
- int ContainsNode (UEdGraphNode ∗InNode)

  Checks that the provided Node is Indexed, and retrieves its Key.
- void ContainsNode (UEdGraphNode ∗InNode, int &OutIndex)

  Checks that the provided Node is Indexed, and retrieves its Key.
- int GetKey (const UEdGraphNode ∗InNode)

  Gets the Key Linked to the Provided Node in the Indexer.
- bool GetKey (const UEdGraphNode ∗InNode, int &OutKey)

  Gets the Key Linked to the Provided Node in the Indexer.
- void GetNode (const int &InIndex, UEdGraphNode ∗OutNode)

  Gets the Node Linked to the Provided Index.
- UEdGraphNode ∗ GetNode (const int &InIndex)

  Gets the Node Linked to the Provided Index.

- void GetPin (const int &InNodeIndex, const int &InPinIndex, UEdGraphPin ∗OutPin)

  Gets the Pin Linked to the Provided Index, of the Provided Node Index.
- UEdGraphPin ∗ GetPin (const int &InNodeIndex, const int &InPinIndex)

  Gets the Pin Linked to the Provided Index, of the Provided Node Index.
- int AddNode (const UEdGraphNode ∗Node)

  Adds the Given Node to the Indexer.
- void AddNode (int &OutIndex, const UEdGraphNode &InNode)

  Adds the Given Node to the Indexer.
- int GetOrAddNode (const UEdGraphNode ∗InNode)

  Gets or Adds the Provided Node to the Indexer.
- void GetOrAddNode (const UEdGraphNode ∗InNode, int &OutIndex)

  Gets or Adds the Provided Node to the Indexer.
- void RemoveNode (const int &InIndex)

  Removes the Node from the Indexer, linked to the Provided Index.
- void RemoveNode (const UEdGraphNode ∗InNode)

  Removes the Node from the Indexer, finds the Index in the Process.
- void OnGraphEvent (const FEdGraphEditAction &InAction)

  Callback for when the Linked Graph for the Indexer has been Modified.
- void OnGraphRebuild ()

  Calls a Full Rebuild of the Indexer, to ensure all Nodes are Indexed.

Protected Attributes

- TMap< int, UEdGraphNode ∗ > IndexMap

  Map of the Index to the Node.
- TSet< int32 > NodeSet

  Look-Up Set of the Nodes Contained in the Indexer.
- TQueue< int32 > AvailableIndices

  A Queue of the Available Indicies for the Indexer, that was previously in use but made vacant.
- UEdGraph ∗ LinkedGraph

  The Graph Being Indexed By This Indexer.
- FDelegateHandle OnGraphChangedHandle

4.6.1 Detailed Description

Definition at line 14 of file GraphIndexer.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 FGraphIndexer() [1/2]

```
FGraphIndexer::FGraphIndexer ( )
```

Definition at line 12 of file GraphIndexer.cpp.

```
00013 {
00014
00015 }
```

4.6.2.2 FGraphIndexer() [2/2]

```
FGraphIndexer::FGraphIndexer (
            const UEdGraph * GraphToIndex )
```

Definition at line 17 of file GraphIndexer.cpp.

```
    : LinkedGraph(const_cast<UEdGraph*>(GraphToIndex))
{
    BuildGraphIndex();

    OnGraphChangedHandle = LinkedGraph->AddOnGraphChangedHandler(
        FOnGraphChanged::FDelegate::CreateRaw(this, &FGraphIndexer::OnGraphEvent)
    );
}
```

4.6.2.3 ∼FGraphIndexer()

```
FGraphIndexer::∼FGraphIndexer ( )
```

Definition at line 27 of file GraphIndexer.cpp.

```
{
    IndexMap.Empty();
    NodeSet.Empty();
    AvailableIndices.Empty();

    LinkedGraph->RemoveOnGraphChangedHandler(OnGraphChangedHandle);

    LinkedGraph = nullptr;
}
```

4.6.3 Member Function Documentation

4.6.3.1 AddNode() [1/2]

```
int FGraphIndexer::AddNode (
            const UEdGraphNode * Node )
```

Adds the Given Node to the Indexer.

Parameters

Node	The Node To Add To The Indexer.

Returns

The Index of the Node in the Indexer.

Definition at line 134 of file GraphIndexer.cpp.

```
{
    check (InNode != nullptr);

    if (!InNode->IsValidLowLevelFast())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided Node is not valid."))
    }

    int Index = ContainsNode(const_cast<UEdGraphNode*>(InNode));
    if (Index != -1)
    {
        return Index;
    }

    GetAvailableIndex(Index);

    NodeSet.Add(InNode->GetUniqueID());
    IndexMap.Add(Index, const_cast<UEdGraphNode*>(InNode));

    return Index;
}
```

4.6.3.2 AddNode() [2/2]

```
void FGraphIndexer::AddNode (
            int & OutIndex,
            const UEdGraphNode & InNode )
```

Adds the Given Node to the Indexer.

Parameters

OutIndex	The Index of the Node in the Indexer.
InNode	The Node To Add to the Indexer.

Definition at line 157 of file GraphIndexer.cpp.

```
{
    OutIndex = AddNode(&InNode);
}
```

4.6.3.3 ContainsKey()

```
bool FGraphIndexer::ContainsKey (
            const int & InKey )
```

Checks if the Provided Key is Contained within the Indexer.

Parameters

InKey	The Key To Check if used in the Indexer.

Returns

True, if the Key is Used for Indexing. False, if the Key is Not Used for Indexing.

Definition at line 38 of file GraphIndexer.cpp.

```
{
    return IndexMap.Contains(InKey);
}
```

4.6.3.4 ContainsNode() [1/2]

```
int FGraphIndexer::ContainsNode (
            UEdGraphNode * InNode )
```

Checks that the provided Node is Indexed, and retrieves its Key.

Parameters

InNode	The Node to Find.

Returns

The Key Used to Index The Provided Node. -1 if Unsuccessful in finding the Node.

Definition at line 43 of file GraphIndexer.cpp.

```
{
    check(InNode != nullptr);

    if (!InNode->IsValidLowLevelFast() || !NodeSet.Contains(InNode->GetUniqueID()))
        return -1;

    const int* ReturnedIndex = IndexMap.FindKey(InNode);

    if (ReturnedIndex != nullptr)
    {
        return *ReturnedIndex;
    }
    else return -1;
}
```

4.6.3.5 ContainsNode() [2/2]

```
void FGraphIndexer::ContainsNode (
            UEdGraphNode * InNode,
            int & OutIndex )
```

Checks that the provided Node is Indexed, and retrieves its Key.

Parameters

InNode	The Node to Find.
OutIndex	The Index Linked to the Provided Node.

Definition at line 59 of file GraphIndexer.cpp.

```
{
    OutIndex = ContainsNode(InNode);
}
```

4.6.3.6 GetKey() [1/2]

```
int FGraphIndexer::GetKey (
            const UEdGraphNode * InNode )
```

Gets the Key Linked to the Provided Node in the Indexer.

Parameters

InNode	The Node to find the Index of.

Returns

The Index of the Provided Node. -1 on Failure.

Definition at line 64 of file GraphIndexer.cpp.

```
{
    check(InNode != nullptr);

    if (!InNode->IsValidLowLevelFast())
        return -1;

    const int* FoundKey = IndexMap.FindKey(const_cast<UEdGraphNode*>(InNode));

    if (FoundKey != nullptr) return *FoundKey;
    else return -1;
}
```

4.6.3.7 GetKey() [2/2]

```
bool FGraphIndexer::GetKey (
            const UEdGraphNode * InNode,
            int & OutKey )
```

Gets the Key Linked to the Provided Node in the Indexer.

Parameters

InNode	The Node to find the Index of.
OutKey	The Index of the Provided Node.

Returns

True, if the Key Was Found. False, if the Key Could Not Be Found.

Definition at line 77 of file GraphIndexer.cpp.

```
{
    check(InNode != nullptr);

    if (!InNode->IsValidLowLevelFast())
        return false;

    const int* FoundKey = IndexMap.FindKey(const_cast<UEdGraphNode*>(InNode));
    if (FoundKey != nullptr)
    {
        OutKey = *FoundKey;
        return true;
    }
    else return false;
}
```

4.6.3.8 GetNode() [1/2]

```
UEdGraphNode * FGraphIndexer::GetNode (
              const int & InIndex )
```

Gets the Node Linked to the Provided Index.

Parameters

InIndex	The Index to Find the Node of.

Returns

The Found Graph Node, nullptr on Failure.

Definition at line 93 of file GraphIndexer.cpp.

```
{
    if (!IndexMap.Contains(InIndex))
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided Index is not recognised"))

        return nullptr;
    }

    return IndexMap[InIndex];
}
```

4.6.3.9 GetNode() [2/2]

```
void FGraphIndexer::GetNode (
              const int & InIndex,
              UEdGraphNode * OutNode )
```

Gets the Node Linked to the Provided Index.

Parameters

InIndex	The Index to Find the Node of.
OutNode	Applies the Found Node, else nullptr.

Definition at line 129 of file GraphIndexer.cpp.

```
{
    OutNode = GetNode(InIndex);
}
```

4.6.3.10 GetOrAddNode() [1/2]

```
int FGraphIndexer::GetOrAddNode (
              const UEdGraphNode * InNode )
```

Gets or Adds the Provided Node to the Indexer.

Parameters

InNode	The Node to Get or Look-Up in the Indexer.

Returns

The Index of the Node in the Graph.

Definition at line 162 of file GraphIndexer.cpp.

```
{
    int Key = GetKey(InNode);
    if (Key != -1)
    {
        return Key;
    }

    return AddNode(InNode);
}
```

4.6.3.11 GetOrAddNode() [2/2]

```
void FGraphIndexer::GetOrAddNode (
             const UEdGraphNode * InNode,
             int & OutIndex )
```

Gets or Adds the Provided Node to the Indexer.

Parameters

InNode	The Node to Get or Look-Up in the Indexer.
OutIndex	The Index of the Node in the Graph.

Definition at line 173 of file GraphIndexer.cpp.

```
{
    OutIndex = GetKey(InNode);
    if (OutIndex != -1)
    {
        return;
    }

    OutIndex = AddNode(InNode);
}
```

4.6.3.12 GetPin() [1/2]

```
UEdGraphPin * FGraphIndexer::GetPin (
             const int & InNodeIndex,
             const int & InPinIndex )
```

Gets the Pin Linked to the Provided Index, of the Provided Node Index.

Parameters

InNodeIndex	The Index of the Node to find the Pin From.
InPinIndex	The Index of the Pin on the Provided Node.

Returns

The Found Pin on the Provided Node, nullptr on Failure.

Definition at line 117 of file GraphIndexer.cpp.

```
{
    UEdGraphNode* Node = GetNode(InNodeIndex);
    if (Node == nullptr)
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Requested Node at index %d is not valid."),
      InNodeIndex);
        return nullptr;
    }

    return Node->GetPinAt(InPinIndex); // Returns nullptr if invalid
}
```

4.6.3.13 GetPin() [2/2]

```
void FGraphIndexer::GetPin (
            const int & InNodeIndex,
            const int & InPinIndex,
            UEdGraphPin * OutPin )
```

Gets the Pin Linked to the Provided Index, of the Provided Node Index.

Parameters

InNodeIndex	The Index of the Node to find the Pin From.
InPinIndex	The Index of the Pin on the Provided Node.
OutPin	The Found Pin on the Provided Node.

Definition at line 105 of file GraphIndexer.cpp.

```
{
    UEdGraphNode* Node = GetNode(InNodeIndex);
    if (Node == nullptr)
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Requested Node at index %d is not valid."),
      InNodeIndex);
        return;
    }

    OutPin = Node->GetPinAt(InPinIndex); // Returns nullptr if invalid
}
```

4.6.3.14 OnGraphEvent()

```
void FGraphIndexer::OnGraphEvent (
            const FEdGraphEditAction & InAction )
```

Callback for when the Linked Graph for the Indexer has been Modified.

Parameters

InAction	

Definition at line 225 of file GraphIndexer.cpp.

```
{
    if (InAction.Graph != LinkedGraph)
    {
        return;
    }

    switch (InAction.Action)
    {
        case EEdGraphActionType::GRAPHACTION_AddNode:
        {
            for (const UEdGraphNode* Node : InAction.Nodes)
            {
                AddNode(Node);
            }

            break;
        }

        case EEdGraphActionType::GRAPHACTION_RemoveNode:
        {
            for (const UEdGraphNode* Node : InAction.Nodes)
            {
                RemoveNode(Node);
            }

            break;
        }
    }
}
```

4.6.3.15 OnGraphRebuild()

```
void FGraphIndexer::OnGraphRebuild ( )
```

Calls a Full Rebuild of the Indexer, to ensure all Nodes are Indexed.

Definition at line 256 of file GraphIndexer.cpp.

```
{
    IndexMap.Reset();
    NodeSet.Reset();
    AvailableIndices.Empty();

    BuildGraphIndex();
}
```

4.6.3.16 RemoveNode() [1/2]

```
void FGraphIndexer::RemoveNode (
            const int & InIndex )
```

Removes the Node from the Indexer, linked to the Provided Index.

Parameters

InIndex	The Index to Remove from the Indexer, and its Linked Node.

Definition at line 184 of file GraphIndexer.cpp.

```
{
    if (!IndexMap.Contains(InIndex))
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided Index is not recognised"))
    }

    const UEdGraphNode* Node = IndexMap[InIndex];

    if (Node->IsValidLowLevelFast())
    {
        NodeSet.Remove(Node->GetUniqueID());
        IndexMap.Remove(InIndex);
        AvailableIndices.Enqueue(InIndex);
    }
    else
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Stored Node in IndexMap is not vaild."))
    }
}
```

4.6.3.17 RemoveNode() [2/2]

```
void FGraphIndexer::RemoveNode (
              const UEdGraphNode * InNode )
```

Removes the Node from the Indexer, finds the Index in the Process.

Parameters

InNode	The Node To Remove from the Indexer, and its Linked Index.

Definition at line 205 of file GraphIndexer.cpp.

```
{
    check(InNode != nullptr);

    int Key = GetKey(InNode);
    if (Key == -1)
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Node does not exist in IndexMap."))
        return;
    }

    RemoveNode(Key);
}
```

4.6.4 Member Data Documentation

4.6.4.1 AvailableIndices

```
TQueue<int32> FGraphIndexer::AvailableIndices  [protected]
```

A Queue of the Available Indicies for the Indexer, that was previously in use but made vacant.

Definition at line 173 of file GraphIndexer.h.

4.6.4.2 IndexMap

```
TMap<int, UEdGraphNode*> FGraphIndexer::IndexMap  [protected]
```

Map of the Index to the Node.

Definition at line 163 of file GraphIndexer.h.

4.6.4.3 LinkedGraph

```
UEdGraph* FGraphIndexer::LinkedGraph  [protected]
```

The Graph Being Indexed By This Indexer.

Definition at line 178 of file GraphIndexer.h.

4.6.4.4 NodeSet

```
TSet<int32> FGraphIndexer::NodeSet  [protected]
```

Look-Up Set of the Nodes Contained in the Indexer.

Definition at line 168 of file GraphIndexer.h.

4.6.4.5 OnGraphChangedHandle

```
FDelegateHandle FGraphIndexer::OnGraphChangedHandle  [protected]
```

Definition at line 180 of file GraphIndexer.h.

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

- Source/OpenAccessibility/Public/GraphIndexer.h
- Source/OpenAccessibility/Private/GraphIndexer.cpp

4.7 FGraphLocomotionChunk Struct Reference

Public Member Functions

- void SetChunkBounds (FVector2D InTopLeft, FVector2D InBottomRight)
- void GetChunkBounds (FVector2D &OutTopLeft, FVector2D &OutBottomRight) const
- FVector2D GetChunkTopLeft () const
- FVector2D GetChunkBottomRight () const
- void SetVisColor (const FLinearColor &NewColor) const

Public Attributes

- FVector2D TopLeft

    Visual Chunks Top Left Corner.
- FVector2D BottomRight

    Visual Chunks Bottom Right Corner.
- TWeakPtr< SBox > ChunkWidget

    Weak Pointer to the Chunks Visual Widget.
- TWeakPtr< SBorder > ChunkVisWidget

    Weak Pointer to the Chunks Visual Widget.
- TWeakPtr< class SIndexer > ChunkIndexer

    Weak Pointer to the Chunks Indexer Widget.

4.7.1 Detailed Description

Definition at line 13 of file AccessibilityGraphLocomotionContext.h.

4.7.2 Member Function Documentation

4.7.2.1 GetChunkBottomRight()

```
FVector2D FGraphLocomotionChunk::GetChunkBottomRight ( ) const  [inline]
```

Definition at line 33 of file AccessibilityGraphLocomotionContext.h.

```
{ return BottomRight; }
```

4.7.2.2 GetChunkBounds()

```
void FGraphLocomotionChunk::GetChunkBounds (
              FVector2D & OutTopLeft,
              FVector2D & OutBottomRight ) const  [inline]
```

Definition at line 25 of file AccessibilityGraphLocomotionContext.h.

```
    {
        OutTopLeft = TopLeft;
        OutBottomRight = BottomRight;
    }
```

4.7.2.3 GetChunkTopLeft()

```
FVector2D FGraphLocomotionChunk::GetChunkTopLeft ( ) const  [inline]
```

Definition at line 31 of file AccessibilityGraphLocomotionContext.h.

```
{ return TopLeft; }
```

4.7.2.4 SetChunkBounds()

```
void FGraphLocomotionChunk::SetChunkBounds (
            FVector2D InTopLeft,
            FVector2D InBottomRight ) [inline]
```

Definition at line 19 of file AccessibilityGraphLocomotionContext.h.

```
    {
        TopLeft = InTopLeft;
        BottomRight = InBottomRight;
    }
```

4.7.2.5 SetVisColor()

```
void FGraphLocomotionChunk::SetVisColor (
            const FLinearColor & NewColor ) const [inline]
```

Definition at line 35 of file AccessibilityGraphLocomotionContext.h.

```
    {
        if (ChunkVisWidget.IsValid())
            ChunkVisWidget.Pin()->SetBorderBackgroundColor(NewColor);
    }
```

4.7.3 Member Data Documentation

4.7.3.1 BottomRight

```
FVector2D FGraphLocomotionChunk::BottomRight
```

Visual Chunks Bottom Right Corner.

Definition at line 51 of file AccessibilityGraphLocomotionContext.h.

4.7.3.2 ChunkIndexer

```
TWeakPtr<class SIndexer> FGraphLocomotionChunk::ChunkIndexer
```

Weak Pointer to the Chunks Indexer Widget.

Definition at line 66 of file AccessibilityGraphLocomotionContext.h.

4.7.3.3 ChunkVisWidget

```
TWeakPtr<SBorder> FGraphLocomotionChunk::ChunkVisWidget
```

Weak Pointer to the Chunks Visual Widget.

Definition at line 61 of file AccessibilityGraphLocomotionContext.h.

4.7.3.4 ChunkWidget

```
TWeakPtr<SBox> FGraphLocomotionChunk::ChunkWidget
```

Weak Pointer to the Chunks Visual Widget.

Definition at line 56 of file AccessibilityGraphLocomotionContext.h.

4.7.3.5 TopLeft

```
FVector2D FGraphLocomotionChunk::TopLeft
```

Visual Chunks Top Left Corner.

Definition at line 46 of file AccessibilityGraphLocomotionContext.h.

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

- Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityGraphLocomotionContext.h

4.8 FIndexer< KeyType, ValueType > Class Template Reference

```
#include <Indexer.h>
```

Public Member Functions

- bool IsEmpty () const
- void Reset ()
- void Empty ()
- int32 Num () const
- void Num (int32 &OutNum) const
- bool ContainsKey (const KeyType &InKey)
- bool ContainsValue (const ValueType &InValue)
- const KeyType GetKey (const ValueType &InValue)
- bool GetKey (const ValueType &InValue, KeyType &OutKey)
- ValueType GetValue (const KeyType &InKey)
- bool GetValue (const KeyType &InKey, ValueType &OutValue)
- KeyType AddValue (const ValueType &InValue)
- void AddValue (const ValueType &InValue, KeyType &OutKey)
- KeyType GetKeyOrAddValue (const ValueType &InValue)
- void GetKeyOrAddValue (const ValueType &InValue, KeyType &OutKey)
- void RemoveValue (const KeyType &InKey)
- void RemoveValue (const ValueType &InValue)

Protected Member Functions

- void GetAvailableKey (KeyType &OutKey)
- KeyType GetAvailableKey ()

Protected Attributes

- TMap< KeyType, ValueType > IndexMap
- TQueue< KeyType > AvailableIndexes

4.8.1 Detailed Description

template<typename KeyType, typename ValueType>
class FIndexer< KeyType, ValueType >

A Templated Indexer for Indexing Assets in a TMap.

Template Parameters

KeyType	Type of the Key Element of the Index.
ValueType	Type of the Value Element of the Index.

Definition at line 15 of file Indexer.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 FIndexer()

```
template<typename KeyType , typename ValueType >
FIndexer< KeyType, ValueType >::FIndexer ( )  [inline]
```

Definition at line 19 of file Indexer.h.

```
00020     {
00021
00022     }
```

4.8.2.2 ∼FIndexer()

```
template<typename KeyType , typename ValueType >
virtual FIndexer< KeyType, ValueType >::∼FIndexer ( )  [inline], [virtual]
```

Definition at line 24 of file Indexer.h.

```
00025     {
00026
00027     }
```

4.8.3 Member Function Documentation

4.8.3.1 AddValue() [1/2]

```
template<typename KeyType , typename ValueType >
KeyType FIndexer< KeyType, ValueType >::AddValue (
              const ValueType & InValue )  [inline]
```

Inserts the specified value into the indexer, and provides its key.

Parameters

InValue	The value to insert.

Returns

The Key of the associated to the inserted value in the indexer.

Definition at line 166 of file Indexer.h.

```
    {
        check(InValue != nullptr);

        if (ContainsValue(InValue))
        {
            return GetKey(InValue);
        }

        KeyType NewKey;
        GetAvailableKey(NewKey);

        IndexMap.Add(NewKey, InValue);

        return NewKey;
    }
```

4.8.3.2 AddValue() [2/2]

```
template<typename KeyType , typename ValueType >
void FIndexer< KeyType, ValueType >::AddValue (
              const ValueType & InValue,
              KeyType & OutKey )  [inline]
```

Inserts the specified value into the indexer, and provides its key.

Parameters

InValue	The value to insert.
OutKey	The Key of the associated to the newly inserted value.

Definition at line 188 of file Indexer.h.

```
    {
        check(InValue != nullptr);

        if (ContainsValue(InValue))
        {
            OutKey = GetKey(InValue);
            return;
        }

        OutKey = GetAvailableKey();

        IndexMap.Add(OutKey, InValue);
    }
```

4.8.3.3 ContainsKey()

```
template<typename KeyType , typename ValueType >
bool FIndexer< KeyType, ValueType >::ContainsKey (
             const KeyType & InKey )  [inline]
```

Checks if the indexer contains the specified key.

Parameters

InKey	The Key to Search For.

Returns

True if the Key is in use in the Indexer, otherwise False.

Definition at line 80 of file Indexer.h.

```
    {
        return IndexMap.Contains(InKey);
    }
```

4.8.3.4 ContainsValue()

```
template<typename KeyType , typename ValueType >
bool FIndexer< KeyType, ValueType >::ContainsValue (
             const ValueType & InValue )  [inline]
```

Checks if the Indexer contains the specified value.

Parameters

InValue	The Value to Search For.

Returns

True of the specified value is associated with the Indexer.

Definition at line 90 of file Indexer.h.

```
    {
        check(InValue != nullptr);

        const KeyType* FoundKey = IndexMap.FindKey(InValue);

        return FoundKey != nullptr;
    }
```

4.8.3.5 Empty()

```
template<typename KeyType , typename ValueType >
void FIndexer< KeyType, ValueType >::Empty ( )  [inline]
```

Empties the Indexer, preserving no space allocated.

Definition at line 51 of file Indexer.h.

```
    {
        IndexMap.Empty();
        AvailableIndexes.Empty();
    }
```

4.8.3.6 GetAvailableKey() `[1/2]`

```
template<typename KeyType , typename ValueType >
KeyType FIndexer< KeyType, ValueType >::GetAvailableKey ( )  [inline], [protected]
```

Gets the Next Available Key in the Indexer.

Returns

The next available key in the indexer.

Definition at line 285 of file Indexer.h.

```
    {
        if (!AvailableIndexes.IsEmpty())
        {
            KeyType OutKey;
            if (AvailableIndexes.Dequeue(OutKey))
                return OutKey;
        }

        return IndexMap.Num();
    }
```

4.8.3.7 GetAvailableKey() `[2/2]`

```
template<typename KeyType , typename ValueType >
void FIndexer< KeyType, ValueType >::GetAvailableKey (
             KeyType & OutKey )  [inline], [protected]
```

Gets the Next Available Key in the Indexer.

Parameters

OutKey	Sets the Next Available Key.

Definition at line 273 of file Indexer.h.

```
    {
        if (!AvailableIndexes.IsEmpty() && AvailableIndexes.Dequeue(OutKey))
            return;

        OutKey = IndexMap.Num();
    }
```

4.8.3.8 GetKey() [1/2]

```
template<typename KeyType , typename ValueType >
const KeyType FIndexer< KeyType, ValueType >::GetKey (
              const ValueType & InValue )  [inline]
```

Gets the associated Key with the specified value.

Parameters

InValue	The value to search using.

Returns

The associated key for the specified value.

Definition at line 104 of file Indexer.h.

```
    {
        check(InValue != nullptr);

        return *IndexMap.FindKey(InValue);
    }
```

4.8.3.9 GetKey() [2/2]

```
template<typename KeyType , typename ValueType >
bool FIndexer< KeyType, ValueType >::GetKey (
              const ValueType & InValue,
              KeyType & OutKey )  [inline]
```

Gets the associated Key with the specified value.

Parameters

InValue	The value to search using.
OutKey	Sets the associated key for the specified value

Returns

True if the associated key was found, otherwise False.

Definition at line 117 of file Indexer.h.

```
    {
        check(InValue != nullptr);

        const KeyType* FoundKey = IndexMap.FindKey(InValue);

        if (FoundKey != nullptr)
        {
            OutKey = *FoundKey;

            return true;
        }
        else return false;
    }
```

4.8.3.10 GetKeyOrAddValue() [1/2]

```
template<typename KeyType , typename ValueType >
KeyType FIndexer< KeyType, ValueType >::GetKeyOrAddValue (
            const ValueType & InValue )  [inline]
```

Finds or Inserts the specified value into the Indexer.

Parameters

InValue	The value to find or insert into the indexer.

Returns

The Key of the associated value.

Definition at line 208 of file Indexer.h.

```
    {
        check(InValue != nullptr);

        KeyType FoundKey;
        if (GetKey(InValue, FoundKey))
            return FoundKey;

        return AddValue(InValue);
    }
```

4.8.3.11 GetKeyOrAddValue() [2/2]

```
template<typename KeyType , typename ValueType >
void FIndexer< KeyType, ValueType >::GetKeyOrAddValue (
            const ValueType & InValue,
            KeyType & OutKey )  [inline]
```

Finds or Inserts the specified value into the Indexer.

Parameters

InValue	The value to find or insert into the indexer.
OutKey	Sets the Key of the associated value.

Definition at line 224 of file Indexer.h.

```
{
    check(InValue != nullptr);

    if (GetKey(InValue, OutKey))
        return;

    OutKey = AddValue(InValue);
}
```

4.8.3.12 GetValue() [1/2]

```
template<typename KeyType , typename ValueType >
ValueType FIndexer< KeyType, ValueType >::GetValue (
              const KeyType & InKey )  [inline]
```

Gets the value linked to the specified key.

Parameters

InKey	The Key to Search using.

Returns

The associated value of the specified key.

Definition at line 137 of file Indexer.h.

```
{
    return *IndexMap.Find(InKey);
}
```

4.8.3.13 GetValue() [2/2]

```
template<typename KeyType , typename ValueType >
bool FIndexer< KeyType, ValueType >::GetValue (
              const KeyType & InKey,
              ValueType & OutValue )  [inline]
```

Gets the value linked to the specified key.

Parameters

InKey	The Key to Search using.
OutValue	Sets the associated value of the specified key.

Returns

True if an associated value was found, otherwise False.

Definition at line 148 of file Indexer.h.

```
    {
        if (!IndexMap.Contains(InKey))
        {
            UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided Key is not recognised."));
            return false;
        }

        OutValue = *IndexMap.Find(InKey);

        return true;
    }
```

4.8.3.14 IsEmpty()

```
template<typename KeyType , typename ValueType >
bool FIndexer< KeyType, ValueType >::IsEmpty ( ) const  [inline]
```

Checks if the Indexer is Empty.

Returns

True if the Indexer is Empty, otherwise False.

Definition at line 34 of file Indexer.h.

```
    {
        return IndexMap.IsEmpty();
    }
```

4.8.3.15 Num() [1/2]

```
template<typename KeyType , typename ValueType >
int32 FIndexer< KeyType, ValueType >::Num ( ) const  [inline]
```

Gets the Number of Items Currently in the Indexer.

Returns

Number of Items being Indexed.

Definition at line 61 of file Indexer.h.

```
    {
        return IndexMap.Num();
    }
```

4.8.3.16 Num() [2/2]

```
template<typename KeyType , typename ValueType >
void FIndexer< KeyType, ValueType >::Num (
            int32 & OutNum ) const  [inline]
```

Gets the Number of Items Currently in the Indexer.

Parameters

OutNum	Sets to the Number of Items Being Indexed.

Definition at line 70 of file Indexer.h.

```
    {
        OutNum = IndexMap.Num();
    }
```

4.8.3.17 RemoveValue() [1/2]

```
template<typename KeyType , typename ValueType >
void FIndexer< KeyType, ValueType >::RemoveValue (
            const KeyType & InKey ) [inline]
```

Removes the specified key from the Indexer.

Parameters

InKey	The key to remove from the indexer.

Definition at line 238 of file Indexer.h.

```
    {
        if (!IndexMap.Contains(InKey))
        {
            UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided Key Has No Pair in Index."));
            return;
        }

        IndexMap.Remove(InKey);
        AvailableIndexes.Enqueue(InKey);
    }
```

4.8.3.18 RemoveValue() [2/2]

```
template<typename KeyType , typename ValueType >
void FIndexer< KeyType, ValueType >::RemoveValue (
            const ValueType & InValue ) [inline]
```

Removes the specified value and its associated key from the Indexer.

Parameters

InValue	The value to remove from the Indexer.

Definition at line 254 of file Indexer.h.

```
    {
        check(InValue != nullptr);

        KeyType FoundKey;
        if (GetKey(InValue, FoundKey))
        {
            IndexMap.Remove(FoundKey);
            AvailableIndexes.Enqueue(FoundKey);
        }
        else UE_LOG(LogOpenAccessibility, Log, TEXT("Provided Value Had No Associated Key."));
    }
```

4.8.3.19 Reset()

```
template<typename KeyType , typename ValueType >
void FIndexer< KeyType, ValueType >::Reset ( )  [inline]
```

Empties the Indexer, but preserves all Allocations.

Definition at line 42 of file Indexer.h.

```
    {
        IndexMap.Reset();
        AvailableIndexes.Empty();
    }
```

4.8.4 Member Data Documentation

4.8.4.1 AvailableIndexes

```
template<typename KeyType , typename ValueType >
TQueue<KeyType> FIndexer< KeyType, ValueType >::AvailableIndexes  [protected]
```

The Queue of Available Indexes from Previous Associations.

Definition at line 310 of file Indexer.h.

4.8.4.2 IndexMap

```
template<typename KeyType , typename ValueType >
TMap<KeyType, ValueType> FIndexer< KeyType, ValueType >::IndexMap  [protected]
```

The Map of Keys to Associated Values.

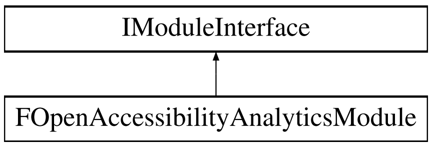
Definition at line 305 of file Indexer.h.

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

- Source/OpenAccessibility/Public/Indexers/Indexer.h

4.9 FOpenAccessibilityAnalyticsModule Class Reference

Inheritance diagram for FOpenAccessibilityAnalyticsModule:



Public Member Functions

- virtual void StartupModule () override
- virtual void ShutdownModule () override
- virtual bool SupportsDynamicReloading () override
- bool DumpTick (float DeltaTime)
- void LogEvent (const TCHAR ∗EventTitle, const TCHAR ∗LogString,...)

Static Public Member Functions

- static FOpenAccessibilityAnalyticsModule & Get ()

4.9.1 Detailed Description

Definition at line 15 of file OpenAccessibilityAnalytics.h.

4.9.2 Member Function Documentation

4.9.2.1 DumpTick()

```
bool FOpenAccessibilityAnalyticsModule::DumpTick (
             float DeltaTime )
```

The Tick Event for Handling the Dump Event.

Parameters

DeltaTime	Time since last Tick.

Returns

Definition at line 23 of file OpenAccessibilityAnalytics.cpp.

```
{
    if (EventBuffer.IsEmpty())
        return true;

    if (SessionBufferFile.IsEmpty())
        SessionBufferFile = GenerateFileForSessionLog();

    UE_LOG(LogOpenAccessibilityAnalytics, Log, TEXT("Dumping Event Log To File."));

    if (!WriteBufferToFile())
    {
        UE_LOG(LogOpenAccessibilityAnalytics, Warning, TEXT("EventLog Dumping Failed."));
    }

    return true;
}
```

4.9.2.2 Get()

`static FOpenAccessibilityAnalyticsModule & FOpenAccessibilityAnalyticsModule::Get ( ) [inline], [static]`

End IModuleInterface Implementation

Definition at line 28 of file OpenAccessibilityAnalytics.h.

```
    {
        return
  FModuleManager::GetModuleChecked<FOpenAccessibilityAnalyticsModule>("OpenAccessibilityAnalytics");
    }
```

4.9.2.3 LogEvent()

```
FORCEINLINE void FOpenAccessibilityAnalyticsModule::LogEvent (
                const TCHAR * EventTitle,
                const TCHAR * LogString,
                 ... )
```

Logs the Event to the Analytics Module.

Parameters

EventTitle	Title of the Log Event.
LogString	Body of the Log Event
...	

Definition at line 135 of file OpenAccessibilityAnalytics.h.

```
{
    va_list Args;

    va_start(Args, LogString);
    TStringBuilder<1024> Message;
    Message.AppendV(LogString, Args);
    va_end(Args);

    EventBuffer.Add(
        LoggedEvent(EventTitle, *Message)
    );
}
```

4.9.2.4 ShutdownModule()

```
void FOpenAccessibilityAnalyticsModule::ShutdownModule ( )  [override], [virtual]
```

Definition at line 17 of file OpenAccessibilityAnalytics.cpp.

```
{
    DisableDumpTick();
    RemoveConsoleCommands();
}
```

4.9.2.5 StartupModule()

```
void FOpenAccessibilityAnalyticsModule::StartupModule ( )  [override], [virtual]
```

IModuleInterface Implementation

Definition at line 9 of file OpenAccessibilityAnalytics.cpp.

```
{
    SessionBufferFile = GenerateFileForSessionLog();

    EnableDumpTick();
    AddConsoleCommands();
}
```

4.9.2.6 SupportsDynamicReloading()

```
virtual bool FOpenAccessibilityAnalyticsModule::SupportsDynamicReloading ( )  [inline], [override],
[virtual]
```

Definition at line 24 of file OpenAccessibilityAnalytics.h.

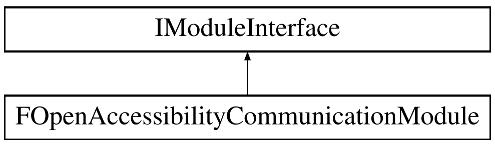
```
00024 { return false; }
```

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

- Source/OpenAccessibilityAnalytics/Public/OpenAccessibilityAnalytics.h
- Source/OpenAccessibilityAnalytics/Private/OpenAccessibilityAnalytics.cpp

4.10 FOpenAccessibilityCommunicationModule Class Reference

Inheritance diagram for FOpenAccessibilityCommunicationModule:



Public Member Functions

- virtual void StartupModule () override
- virtual void ShutdownModule () override
- virtual bool SupportsDynamicReloading () override
- bool Tick (const float DeltaTime)
- void HandleKeyDownEvent (const FKeyEvent &InKeyEvent)
- void TranscribeWaveForm (TArray< float > AudioBufferToTranscribe)

  Sends the Audio Buffer to the Transcription Service.

Static Public Member Functions

- static FOpenAccessibilityCommunicationModule & Get ()

Public Attributes

- TMulticastDelegate< void(TArray< FString >)> OnTranscriptionRecieved

  A Delegate for when Transcriptions are recived back from the Transcription Service.
- class UAudioManager * AudioManager

  The AudioManager, Managing any Audio Capture Component.
- TSharedPtr< class FSocketCommunicationServer > SocketServer

  The Socket Communication Server, Managing Socket Communication for the Transcription Service.
- TSharedPtr< FPhraseTree > PhraseTree

  The PhraseTree, Containing any Bound Phrase Nodes and Commands to Execute from Transcriptions.
- class UPhraseTreeUtils * PhraseTreeUtils

  Phrase Tree Utility Class, For Dealing With Phrase Tree Function Libraries.

4.10.1 Detailed Description

Definition at line 16 of file OpenAccessibilityCommunication.h.

4.10.2 Member Function Documentation

4.10.2.1 Get()

```
static FOpenAccessibilityCommunicationModule & FOpenAccessibilityCommunicationModule::Get ( )
[inline], [static]
```

End IModuleInterface Implementation

Definition at line 31 of file OpenAccessibilityCommunication.h.

```
00032     {
00033         return
      FModuleManager::GetModuleChecked<FOpenAccessibilityCommunicationModule>("OpenAccessibilityCommunication");
00034     }
```

4.10.2.2 HandleKeyDownEvent()

```
void FOpenAccessibilityCommunicationModule::HandleKeyDownEvent (
            const FKeyEvent & InKeyEvent )
```

Definition at line 92 of file OpenAccessibilityCommunication.cpp.

```
{
    // If the Space Key is pressed, we will send a request to the Accessibility Server
    if (InKeyEvent.GetKey() == EKeys::SpaceBar)
    {
        if (InKeyEvent.IsShiftDown())
        {
            OA_LOG(LogOpenAccessibilityCom, Log, TEXT("AudioCapture Change"), TEXT("Stopping Audio
      Capture"));
            AudioManager->StopCapturingAudio();
        }
        else
        {
            OA_LOG(LogOpenAccessibilityCom, Log, TEXT("AudioCapture Change"), TEXT("Starting Audio
      Capture"));
            AudioManager->StartCapturingAudio();
        }
    }
}
```

4.10.2.3 ShutdownModule()

```
void FOpenAccessibilityCommunicationModule::ShutdownModule ( )  [override], [virtual]
```

Definition at line 55 of file OpenAccessibilityCommunication.cpp.

```
{
    // This function may be called during shutdown to clean up your module.  For modules that support
      dynamic reloading,
    // we call this function before unloading the module.
    UE_LOG(LogOpenAccessibilityCom, Display, TEXT("OpenAccessibilityComModule::ShutdownModule()"));

    AudioManager->RemoveFromRoot();
    PhraseTreeUtils->RemoveFromRoot();

    FSlateApplication::Get().OnApplicationPreInputKeyDownListener().Remove(KeyDownEventHandle);

    UnloadZMQDLL();

    UnregisterConsoleCommands();
}
```

4.10.2.4 StartupModule()

```
void FOpenAccessibilityCommunicationModule::StartupModule ( )  [override], [virtual]
```

IModuleInterface Implementation

Definition at line 24 of file OpenAccessibilityCommunication.cpp.

```
{
    LoadZMQDLL();

    // This code will execute after your module is loaded into memory; the exact timing is specified
      in the .uplugin file per-module
    UE_LOG(LogOpenAccessibilityCom, Display, TEXT("OpenAccessibilityComModule::StartupModule()"));

    // Initialize AudioManager
    AudioManager = NewObject<UAudioManager>();
    AudioManager->AddToRoot();

    AudioManager->OnAudioReadyForTranscription
        .BindRaw(this, &FOpenAccessibilityCommunicationModule::TranscribeWaveForm);

    // Initialize Socket Server
    SocketServer = MakeShared<FSocketCommunicationServer>();

    // Build The Phrase Tree
    BuildPhraseTree();

    // Bind Tick Event
    TickDelegate = FTickerDelegate::CreateRaw(this, &FOpenAccessibilityCommunicationModule::Tick);
    TickDelegateHandle = FTSTicker::GetCoreTicker().AddTicker(TickDelegate);

    // Bind Input Events
    KeyDownEventHandle = FSlateApplication::Get().OnApplicationPreInputKeyDownListener().AddRaw(this,
  &FOpenAccessibilityCommunicationModule::HandleKeyDownEvent);

    // Register Console Commands
    RegisterConsoleCommands();
}
```

4.10.2.5 SupportsDynamicReloading()

`virtual bool FOpenAccessibilityCommunicationModule::SupportsDynamicReloading ( ) [inline], [override], [virtual]`

Definition at line 25 of file OpenAccessibilityCommunication.h.

```
    {
        return false;
    }
```

4.10.2.6 Tick()

```
bool FOpenAccessibilityCommunicationModule::Tick (
               const float DeltaTime )
```

Definition at line 71 of file OpenAccessibilityCommunication.cpp.

```
{
    // Detect if any events are ready to be received.
    if (SocketServer->EventOccured())
    {
        TArray<FString> RecvStrings;
        TSharedPtr<FJsonObject> RecvMetadata;

        // Receive the Detected Event, with separate transcriptions and metadata.
        if (SocketServer->RecvStringMultipartWithMeta(RecvStrings, RecvMetadata))
        {
            OA_LOG(LogOpenAccessibilityCom, Log, TEXT("TRANSCRIPTION RECIEVED"), TEXT("Recieved
  Multipart - Message Count: %d"), RecvStrings.Num());

            // Send Received Transcriptions to any bound events.
            OnTranscriptionRecieved.Broadcast(RecvStrings);
        }
    }

    return true;
}
```

4.10.2.7 TranscribeWaveForm()

```
void FOpenAccessibilityCommunicationModule::TranscribeWaveForm (
               TArray< float > AudioBufferToTranscribe )
```

Sends the Audio Buffer to the Transcription Service.

Parameters

AudioBufferToTranscribe	- The Audiobuffer To Send For Transcription.

Definition at line 110 of file OpenAccessibilityCommunication.cpp.

```
{
    if (AudioBufferToTranscribe.Num() == 0)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Transcription Ready || Audio Buffer is Empty ||"));
        return;
    }

    PrevAudioBuffer = TArray(AudioBufferToTranscribe);

    UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| WaveForm Transcription || Array Size: %d || Byte Size: %s ||"), AudioBufferToTranscribe.Num(), *FString::FromInt(AudioBufferToTranscribe.Num() * sizeof(float)));

    // Create Metadata of Audio Source.
    TSharedPtr<FJsonObject> AudioBufferMetadata = MakeShared<FJsonObject>();
    AudioBufferMetadata->SetNumberField(TEXT("sample_rate"), AudioManager->GetAudioCaptureSampleRate());
    AudioBufferMetadata->SetNumberField(TEXT("num_channels"), AudioManager->GetAudioCaptureNumChannels());

    bool bArrayMessageSent = SocketServer->SendArrayMessageWithMeta(AudioBufferToTranscribe, AudioBufferMetadata.ToSharedRef(), ComSendFlags::none);

    OA_LOG(LogOpenAccessibilityCom, Log, TEXT("TRANSCRIPTION SENT"), TEXT("{%s} Send Audiobuffer (float x %d / %d Hz / %d channels)"),
        bArrayMessageSent ? TEXT("Success") : TEXT("Failed"),
        AudioBufferToTranscribe.Num(), AudioManager->GetAudioCaptureSampleRate(), AudioManager->GetAudioCaptureNumChannels());
}
```

4.10.3 Member Data Documentation

4.10.3.1 AudioManager

`class UAudioManager* FOpenAccessibilityCommunicationModule::AudioManager`

The AudioManager, Managing any Audio Capture Component.

Definition at line 82 of file OpenAccessibilityCommunication.h.

4.10.3.2 OnTranscriptionRecieved

`TMulticastDelegate<void(TArray<FString>)> FOpenAccessibilityCommunicationModule::OnTranscription↩ Recieved`

A Delegate for when Transcriptions are recived back from the Transcription Service.

Definition at line 77 of file OpenAccessibilityCommunication.h.

4.10.3.3 PhraseTree

```
TSharedPtr<FPhraseTree> FOpenAccessibilityCommunicationModule::PhraseTree
```

The PhraseTree, Containing any Bound Phrase Nodes and Commands to Execute from Transcriptions.

Definition at line 92 of file OpenAccessibilityCommunication.h.

4.10.3.4 PhraseTreeUtils

```
class UPhraseTreeUtils* FOpenAccessibilityCommunicationModule::PhraseTreeUtils
```

Phrase Tree Utility Class, For Dealing With Phrase Tree Function Libraries.

Definition at line 97 of file OpenAccessibilityCommunication.h.

4.10.3.5 SocketServer

```
TSharedPtr<class FSocketCommunicationServer> FOpenAccessibilityCommunicationModule::Socket↩
Server
```

The Socket Communication Server, Managing Socket Communication for the Transcription Service.

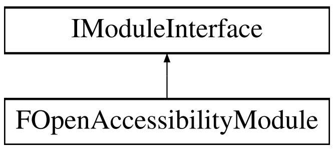
Definition at line 87 of file OpenAccessibilityCommunication.h.

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

- Source/OpenAccessibilityCommunication/Public/OpenAccessibilityCommunication.h
- Source/OpenAccessibilityCommunication/Private/OpenAccessibilityCommunication.cpp

4.11 FOpenAccessibilityModule Class Reference

Inheritance diagram for FOpenAccessibilityModule:



Public Member Functions

- virtual void StartupModule () override
- virtual void ShutdownModule () override
- virtual bool SupportsDynamicReloading () override

Static Public Member Functions

- static FOpenAccessibilityModule & Get ()

Public Attributes

- TSharedPtr< class FAccessibilityNodeFactory > AccessibilityNodeFactory

    The Node Factory for Generating Accessibility Graph Nodes.
- TSharedPtr< class FAssetAccessibilityRegistry > AssetAccessibilityRegistry

    The Registry for Any Asset Accessibility Information.

4.11.1 Detailed Description

Definition at line 11 of file OpenAccessibility.h.

4.11.2 Member Function Documentation

4.11.2.1 Get()

```
static FOpenAccessibilityModule & FOpenAccessibilityModule::Get ( ) [inline], [static]
```

End IModuleInterface Implementation

Definition at line 21 of file OpenAccessibility.h.

```
    {
        return FModuleManager::GetModuleChecked<FOpenAccessibilityModule>("OpenAccessibility");
    }
```

4.11.2.2 ShutdownModule()

```
void FOpenAccessibilityModule::ShutdownModule ( ) [override], [virtual]
```

Definition at line 73 of file OpenAccessibility.cpp.

```
{
    UE_LOG(LogOpenAccessibility, Display, TEXT("OpenAccessibilityModule::ShutdownModule()"));

    UnregisterConsoleCommands();
}
```

4.11.2.3 StartupModule()

`void FOpenAccessibilityModule::StartupModule ( ) [override], [virtual]`

IModuleInterface Implementation

Definition at line 35 of file OpenAccessibility.cpp.

```
{
    UE_LOG(LogOpenAccessibility, Display, TEXT("OpenAccessibilityModule::StartupModule()"));

    // Create the Asset Registry
    AssetAccessibilityRegistry = MakeShared<FAssetAccessibilityRegistry, ESPMode::ThreadSafe>();

    // Register the Accessibility Node Factory
    AccessibilityNodeFactory = MakeShared<FAccessibilityNodeFactory, ESPMode::ThreadSafe>();
    FEdGraphUtilities::RegisterVisualNodeFactory(AccessibilityNodeFactory);

    // Construct Base Phrase Tree Libraries
    FOpenAccessibilityCommunicationModule::Get()
    .PhraseTreeUtils->RegisterFunctionLibrary(
        NewObject<ULocalizedInputLibrary>()
    );

    FOpenAccessibilityCommunicationModule::Get()
    .PhraseTreeUtils->RegisterFunctionLibrary(
        NewObject<UWindowInteractionLibrary>()
    );

    FOpenAccessibilityCommunicationModule::Get()
    .PhraseTreeUtils->RegisterFunctionLibrary(
        NewObject<UViewInteractionLibrary>()
    );

    FOpenAccessibilityCommunicationModule::Get()
    .PhraseTreeUtils->RegisterFunctionLibrary(
        NewObject<UNodeInteractionLibrary>()
    );

    CreateTranscriptionVisualization();

    // Register Console Commands
    RegisterConsoleCommands();
}
```

4.11.2.4 SupportsDynamicReloading()

`virtual bool FOpenAccessibilityModule::SupportsDynamicReloading ( ) [inline], [override], [virtual]`

Definition at line 26 of file OpenAccessibility.h.

```
    {
        return false;
    }
```

4.11.3 Member Data Documentation

4.11.3.1 AccessibilityNodeFactory

`TSharedPtr<class FAccessibilityNodeFactory> FOpenAccessibilityModule::AccessibilityNodeFactory`

The Node Factory for Generating Accessibility Graph Nodes.

Definition at line 81 of file OpenAccessibility.h.

4.11.3.2 AssetAccessibilityRegistry

```
TSharedPtr<class FAssetAccessibilityRegistry> FOpenAccessibilityModule::AssetAccessibility←↩
Registry
```

The Registry for Any Asset Accessibility Information.

Definition at line 86 of file OpenAccessibility.h.

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

- Source/OpenAccessibility/Public/OpenAccessibility.h
- Source/OpenAccessibility/Private/OpenAccessibility.cpp

4.12 FPanelViewPosition Struct Reference

Public Member Functions

- FPanelViewPosition (FVector2D InTopLeft, FVector2D InBotRight)
- bool operator!= (const FVector2D &Other)
- bool operator!= (const FPanelViewPosition &Other)

Public Attributes

- FVector2D TopLeft
- FVector2D BotRight

4.12.1 Detailed Description

Definition at line 70 of file AccessibilityGraphLocomotionContext.h.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 FPanelViewPosition() [1/2]

```
FPanelViewPosition::FPanelViewPosition ( )  [inline]
```

Definition at line 74 of file AccessibilityGraphLocomotionContext.h.

```
            : TopLeft(FVector2D::ZeroVector)
            , BotRight(FVector2D::ZeroVector)
        { }
```

4.12.2.2 FPanelViewPosition() [2/2]

```
FPanelViewPosition::FPanelViewPosition (
            FVector2D InTopLeft,
            FVector2D InBotRight )  [inline]
```

Definition at line 79 of file AccessibilityGraphLocomotionContext.h.

```
        : TopLeft(InTopLeft)
        , BotRight(InBotRight)
    { }
```

4.12.3 Member Function Documentation

4.12.3.1 operator"!=() [1/2]

```
bool FPanelViewPosition::operator!= (
            const FPanelViewPosition & Other )  [inline]
```

Definition at line 89 of file AccessibilityGraphLocomotionContext.h.

```
    {
        return TopLeft != Other.TopLeft || BotRight != Other.BotRight;
    }
```

4.12.3.2 operator"!=() [2/2]

```
bool FPanelViewPosition::operator!= (
            const FVector2D & Other )  [inline]
```

Definition at line 84 of file AccessibilityGraphLocomotionContext.h.

```
    {
        return TopLeft != Other || BotRight != Other;
    }
```

4.12.4 Member Data Documentation

4.12.4.1 BotRight

```
FVector2D FPanelViewPosition::BotRight
```

Definition at line 95 of file AccessibilityGraphLocomotionContext.h.

4.12.4.2 TopLeft

```
FVector2D FPanelViewPosition::TopLeft
```

Definition at line 94 of file AccessibilityGraphLocomotionContext.h.

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

- Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityGraphLocomotionContext.h

4.13 FParseRecord Struct Reference

The Collected Information from the Propogation of the Phrase through the tree.

```
#include <ParseRecord.h>
```

Public Member Functions

- FParseRecord (TArray< UPhraseTreeContextObject ∗ > InContextObjects)
- FString GetPhraseString () const

  Gets the Recorded Phrase String for this record of propagation.
- void AddPhraseString (FString StringToRecord)
- UParseInput ∗ GetPhraseInput (const FString &InString)

  Gets the Input for the Provided Phrase, if it exists.
- template<class CastToType >
  CastToType ∗ GetPhraseInput (const FString &InString)

  Gets the Input for the Provided Phrase, if it exists.
- void GetPhraseInput (const FString &InString, UParseInput ∗OutInput)

  Gets the Input for the Provided Phrase, if it exists.
- template<class CastToType >
  void GetPhraseInput (const FString &InString, CastToType ∗OutInput)

  Gets the Input for the Provided Phrase, if it exists.
- void GetPhraseInputs (const FString &InString, TArray< UParseInput ∗ > &OutInputs, const bool Maintain←Order=true)

  Gets an Array of Phrase Inputs for the Provided Phrase.
- TArray< UParseInput ∗ > GetPhraseInputs (const FString &InString, const bool MaintainOrder=true)

  Gets an Array of Phrase Inputs for the Provided Phrase.
- void AddPhraseInput (const FString &InString, UParseInput ∗InInput)

  Adds a Phrase Input to the Record.
- void RemovePhraseInput (const FString &InString)

  Removes a Phrase Input From The Record.
- void PushContextObj (UPhraseTreeContextObject ∗InObject)

  Pushes a Context Object onto the Stack.
- void PopContextObj ()

  Pops the Top Context Object From The Stack.
- void PopContextObj (UPhraseTreeContextObject ∗OutObject)

  Pops the Top Context Object From The Stack.
- void RemoveContextObj (UPhraseTreeContextObject ∗InObject)

  Removes a Select Context Object From The Stack.

- bool HasContextObj ()

  Checks if there is a Context Object on the Stack.
- bool HasContextObj (UPhraseTreeContextObject ∗InObject)

  Checks if a specific Context Object is on the Stack.
- UPhraseTreeContextObject ∗ GetContextObj ()

  Gets the Top Context Object On The Stack.
- void GetContextObj (UPhraseTreeContextObject ∗OutObject)

  Gets the Top Context Object On The Stack.
- template<class CastToType >
  CastToType ∗ GetContextObj ()

  Gets the Top Context Object On The Stack.
- template<class CastToType >
  void GetContextObj (CastToType ∗OutObject)

  Gets the Top Context Object On The Stack.
- void GetContextStack (TArray< UPhraseTreeContextObject ∗ > OutContextStack)

  Gets the Entire Context Stack.
- TArray< UPhraseTreeContextObject ∗ > GetContextStack ()

  Gets the Entire Context Stack.

Protected Attributes

- TArray< UPhraseTreeContextObject ∗ > ContextObjectStack = TArray<UPhraseTreeContextObject∗>()

  The Context Stack of Context Objects.
- TArray< FString > PhraseRecord

  A Record of the Phrase String used through-out propagation.
- TMultiMap< FString, UParseInput ∗ > PhraseInputs

  Map of all the Provided Phrase Inputs, to their Respective Phrases.

Friends

- class FPhraseTree

4.13.1 Detailed Description

The Collected Information from the Propogation of the Phrase through the tree.

Definition at line 16 of file ParseRecord.h.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 FParseRecord() [1/2]

```
FParseRecord::FParseRecord ( )  [inline]
```

Definition at line 23 of file ParseRecord.h.

```
    {
        PhraseInputs = TMultiMap<FString, UParseInput*>();
        ContextObjectStack = TArray<UPhraseTreeContextObject*>();
    }
```

4.13.2.2 FParseRecord() [2/2]

```
FParseRecord::FParseRecord (
            TArray< UPhraseTreeContextObject * > InContextObjects ) [inline]
```

Definition at line 29 of file ParseRecord.h.

```
{
    PhraseInputs = TMultiMap<FString, UParseInput*>();
    ContextObjectStack = InContextObjects;
}
```

4.13.2.3 ∼FParseRecord()

```
FParseRecord::∼FParseRecord ( ) [inline]
```

Definition at line 35 of file ParseRecord.h.

```
{
    PhraseInputs.Empty();
}
```

4.13.3 Member Function Documentation

4.13.3.1 AddPhraseInput()

```
void FParseRecord::AddPhraseInput (
            const FString & InString,
            UParseInput * InInput ) [inline]
```

Adds a Phrase Input to the Record.

Parameters

InString	- The Phrase to Bind the Input To.
InInput	- The Phrase Input Object Containing Input Data.

Definition at line 162 of file ParseRecord.h.

```
{
    PhraseInputs.Add(InString.ToUpper(), InInput);
}
```

4.13.3.2 AddPhraseString()

```
void FParseRecord::AddPhraseString (
            FString StringToRecord ) [inline]
```

Definition at line 51 of file ParseRecord.h.

```
    {
        PhraseRecord.Add(StringToRecord);
    }
```

4.13.3.3 GetContextObj() [1/4]

`UPhraseTreeContextObject * FParseRecord::GetContextObj ( ) [inline]`

Gets the Top Context Object On The Stack.

Returns

The Top Context Object On The Stack.

Definition at line 249 of file ParseRecord.h.

```
    {
        if (ContextObjectStack.IsEmpty())
            return nullptr;

        return this->ContextObjectStack.Last();
    }
```

4.13.3.4 GetContextObj() [2/4]

```
template<class CastToType >
CastToType * FParseRecord::GetContextObj ( ) [inline]
```

Gets the Top Context Object On The Stack.

Template Parameters

CastToType	DownCast Type For the Context Object (Must Derrive From UPhraseTreeContextObject).

Returns

The DownCasted Context Object, otherwise nullptr.

Definition at line 278 of file ParseRecord.h.

```
    {
        if (ContextObjectStack.IsEmpty())
            return nullptr;

        return Cast<CastToType>(this->ContextObjectStack.Last());
    }
```

4.13.3.5 GetContextObj() [3/4]

```
template<class CastToType >
void FParseRecord::GetContextObj (
            CastToType * OutObject ) [inline]
```

Gets the Top Context Object On The Stack.

Template Parameters

CastToType	DownCast Type For the Context Object (Must Derrive From UPhraseTreeContextObject).

Parameters

OutObject	- Returns the Downcasted Context Object, otherwise nullptr.

Definition at line 292 of file ParseRecord.h.

```
    {
        if (ContextObjectStack.IsEmpty())
        {
            OutObject = nullptr;
            return;
        }

        OutObject = Cast<CastToType>(this->ContextObjectStack.Last());
    }
```

4.13.3.6 GetContextObj() [4/4]

```
void FParseRecord::GetContextObj (
            UPhraseTreeContextObject * OutObject )  [inline]
```

Gets the Top Context Object On The Stack.

Parameters

OutObject	- Returns the Top Context Object On The Stack.

Definition at line 261 of file ParseRecord.h.

```
    {
        if (ContextObjectStack.IsEmpty())
        {
            OutObject = nullptr;
            return;
        }

        OutObject = this->ContextObjectStack.Last();
    }
```

4.13.3.7 GetContextStack() [1/2]

```
TArray< UPhraseTreeContextObject * > FParseRecord::GetContextStack ( )  [inline]
```

Gets the Entire Context Stack.

Returns

The Current Context Stack.

Definition at line 316 of file ParseRecord.h.

```
    {
        return ContextObjectStack;
    }
```

4.13.3.8 GetContextStack() [2/2]

```
void FParseRecord::GetContextStack (
            TArray< UPhraseTreeContextObject * > OutContextStack )  [inline]
```

Gets the Entire Context Stack.

Parameters

OutContextStack	- Returns the Current Context Stack.

Definition at line 307 of file ParseRecord.h.

```
{
    OutContextStack = ContextObjectStack;
}
```

4.13.3.9 GetPhraseInput() [1/4]

```
UParseInput * FParseRecord::GetPhraseInput (
            const FString & InString )  [inline]
```

Gets the Input for the Provided Phrase, if it exists.

Parameters

InString	- The Phrase To Check For An Input.

Returns

The Found PhraseInput For the Phrase, otherwise nullptr.

Definition at line 64 of file ParseRecord.h.

```
{
    // Check If The Phrase Exits
    // This Error Will Be Thrown If: InString Is In Correct (Requires UpperCase) or The Phrase
Does Not Exist.
    check(PhraseInputs.Contains(InString))

    return *PhraseInputs.Find(InString);
}
```

4.13.3.10 GetPhraseInput() [2/4]

```
template<class CastToType >
CastToType * FParseRecord::GetPhraseInput (
            const FString & InString )  [inline]
```

Gets the Input for the Provided Phrase, if it exists.

Template Parameters

CastToType	DownCast Type For the Phrase Input (Must Derrive From UPhraseInput).

Parameters

InString	- The Phrase To Check For An Input.

Returns

The Found DownCasted PhraseInput, otherwise nullptr.

Definition at line 80 of file ParseRecord.h.

```
    {
        // Check If The Phrase Exits
        // This Error Will Be Thrown If: InString Is In Correct (Requires UpperCase) or The Phrase
Does Not Exist.
        check(PhraseInputs.Contains(InString))

        return Cast<CastToType>(*PhraseInputs.Find(InString));
    }
```

4.13.3.11 GetPhraseInput() [3/4]

```
template<class CastToType >
void FParseRecord::GetPhraseInput (
            const FString & InString,
            CastToType * OutInput ) [inline]
```

Gets the Input for the Provided Phrase, if it exists.

Template Parameters

CastToType	DownCast Type For the Phrase Input (Must Derrive From UPhraseInput).

Parameters

InString	- The Phrase To Check For An Input.
OutInput	- Returns the Found DownCasted Input or nullptr.

Definition at line 110 of file ParseRecord.h.

```
    {
        // Check If The Phrase Exits
        // This Error Will Be Thrown If: InString Is In Correct (Requires UpperCase) or The Phrase
Does Not Exist.
        check(PhraseInputs.Contains(InString))

        OutInput = Cast<CastToType>(*PhraseInputs.Find(InString));
    }
```

4.13.3.12 GetPhraseInput() [4/4]

```
void FParseRecord::GetPhraseInput (
            const FString & InString,
            UParseInput * OutInput )  [inline]
```

Gets the Input for the Provided Phrase, if it exists.

Parameters

InString	- The Phrase To Check For An Input.
OutInput	- Returns the Found Input or nullptr.

Definition at line 94 of file ParseRecord.h.

```
    {
        // Check If The Phrase Exits
        // This Error Will Be Thrown If: InString Is In Correct (Requires UpperCase) or The Phrase
 Does Not Exist.
        check(PhraseInputs.Contains(InString))

        OutInput = *PhraseInputs.Find(InString);
    }
```

4.13.3.13 GetPhraseInputs() [1/2]

```
TArray< UParseInput * > FParseRecord::GetPhraseInputs (
            const FString & InString,
            const bool MaintainOrder = true )  [inline]
```

Gets an Array of Phrase Inputs for the Provided Phrase.

Parameters

InString	- The Phrase To Check For A Multi-Input.
MaintainOrder	- Should the Returned Array Maintain the Order the Inputs where Inserted.

Returns

The Array of Found Inputs.

Definition at line 142 of file ParseRecord.h.

```
    {
        // Check If The Phrase Exits
        // This Error Will Be Thrown If: InString Is In Correct (Requires UpperCase) or The Phrase
 Does Not Exist.
        check(PhraseInputs.Contains(InString))

        TArray<UParseInput*> OutInputs;

        PhraseInputs.MultiFind(InString, OutInputs, MaintainOrder);

        return OutInputs;
    }
```

4.13.3.14 GetPhraseInputs() [2/2]

```
void FParseRecord::GetPhraseInputs (
            const FString & InString,
            TArray< UParseInput * > & OutInputs,
            const bool MaintainOrder = true ) [inline]
```

Gets an Array of Phrase Inputs for the Provided Phrase.

Parameters

InString	- The Phrase To Check For A Multi-Input.
OutInputs	- Returns An Array of Inputs.
MaintainOrder	- Should the Returned Array Maintain the Order the Inputs where Inserted.

Definition at line 127 of file ParseRecord.h.

```
    {
        // Check If The Phrase Exits
        // This Error Will Be Thrown If: InString Is In Correct (Requires UpperCase) or The Phrase
      Does Not Exist.
        check(PhraseInputs.Contains(InString))

        PhraseInputs.MultiFind(InString, OutInputs, MaintainOrder);
    }
```

4.13.3.15 GetPhraseString()

```
FString FParseRecord::GetPhraseString ( ) const [inline]
```

Gets the Recorded Phrase String for this record of propagation.

Returns

Definition at line 46 of file ParseRecord.h.

```
    {
        return FString::Join(PhraseRecord, TEXT(" "));
    }
```

4.13.3.16 HasContextObj() [1/2]

```
bool FParseRecord::HasContextObj ( ) [inline]
```

Checks if there is a Context Object on the Stack.

Returns

Definition at line 228 of file ParseRecord.h.

```
    {
        return this->ContextObjectStack.Num() > 0;
    }
```

4.13.3.17 HasContextObj() [2/2]

```
bool FParseRecord::HasContextObj (
            UPhraseTreeContextObject * InObject )  [inline]
```

Checks if a specific Context Object is on the Stack.

Parameters

InObject	- The Context Object To Check if On The Stack.

Returns

True, if the Object is on the Stack. False, if the Object is not on the stack.

Definition at line 238 of file ParseRecord.h.

```
    {
        return HasContextObj() && this->ContextObjectStack.Contains(InObject);
    }
```

4.13.3.18 PopContextObj() [1/2]

```
void FParseRecord::PopContextObj ( )  [inline]
```

Pops the Top Context Object From The Stack.

Definition at line 190 of file ParseRecord.h.

```
    {
        if (ContextObjectStack.IsEmpty())
            return;

        this->ContextObjectStack.Pop();
    }
```

4.13.3.19 PopContextObj() [2/2]

```
void FParseRecord::PopContextObj (
            UPhraseTreeContextObject * OutObject )  [inline]
```

Pops the Top Context Object From The Stack.

Parameters

OutObject	- The Popped Context Object.

Definition at line 202 of file ParseRecord.h.

```
    {
        if (ContextObjectStack.IsEmpty())
        {
            OutObject = nullptr;
            return;
        }

        OutObject = this->ContextObjectStack.Pop();
    }
```

4.13.3.20 PushContextObj()

```
void FParseRecord::PushContextObj (
            UPhraseTreeContextObject * InObject )  [inline]
```

Pushes a Context Object onto the Stack.

Parameters

InObject	- The Context Object To Push onto The Stack.

Definition at line 182 of file ParseRecord.h.

```
    {
        this->ContextObjectStack.Push(InObject);
    }
```

4.13.3.21 RemoveContextObj()

```
void FParseRecord::RemoveContextObj (
            UPhraseTreeContextObject * InObject )  [inline]
```

Removes a Select Context Object From The Stack.

Parameters

InObject	

Definition at line 217 of file ParseRecord.h.

```
    {
        this->ContextObjectStack.Remove(InObject);
    }
```

4.13.3.22 RemovePhraseInput()

```
void FParseRecord::RemovePhraseInput (
            const FString & InString )  [inline]
```

Removes a Phrase Input From The Record.

Parameters

InString	- The Phrase To Remove All Bound Inputs from.

Definition at line 171 of file ParseRecord.h.

```
    {
        PhraseInputs.Remove(InString);
    }
```

4.13.4 Friends And Related Function Documentation

4.13.4.1 FPhraseTree

`friend class FPhraseTree [friend]`

Definition at line 21 of file ParseRecord.h.

4.13.5 Member Data Documentation

4.13.5.1 ContextObjectStack

`TArray<UPhraseTreeContextObject*> FParseRecord::ContextObjectStack = TArray<UPhraseTreeContextObject*>() [protected]`

The Context Stack of Context Objects.

Definition at line 326 of file ParseRecord.h.

4.13.5.2 PhraseInputs

`TMultiMap<FString, UParseInput*> FParseRecord::PhraseInputs [protected]`

Map of all the Provided Phrase Inputs, to their Respective Phrases.

Definition at line 336 of file ParseRecord.h.

4.13.5.3 PhraseRecord

```
TArray<FString> FParseRecord::PhraseRecord [protected]
```

A Record of the Phrase String used through-out propagation.

Definition at line 331 of file ParseRecord.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/ParseRecord.h

4.14 FParseResult Struct Reference

Contains the Result of Propagation through the Phrase Tree.

```
#include <ParseResult.h>
```

Public Member Functions

- FParseResult (PhrasePropogationType InResult)
- FParseResult (PhrasePropogationType InResult, TSharedPtr< FPhraseNode > InReachedNode)

Public Attributes

- uint8_t Result

  The Result of the Propogation.
- TSharedPtr< FPhraseNode > ReachedNode

  The Node that was reached in the tree.

4.14.1 Detailed Description

Contains the Result of Propagation through the Phrase Tree.

Definition at line 51 of file ParseResult.h.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 FParseResult() [1/3]

```
FParseResult::FParseResult ( ) [inline]
```

Definition at line 53 of file ParseResult.h.

```
00054     {
00055         Result = PHRASE_NOT_PARSED;
00056     }
```

4.14.2.2 FParseResult() [2/3]

```
FParseResult::FParseResult (
            PhrasePropogationType InResult ) [inline]
```

Definition at line 58 of file ParseResult.h.

```
{
    Result = InResult;
}
```

4.14.2.3 FParseResult() [3/3]

```
FParseResult::FParseResult (
            PhrasePropogationType InResult,
            TSharedPtr< FPhraseNode > InReachedNode ) [inline]
```

Definition at line 63 of file ParseResult.h.

```
{
    Result = InResult;
    ReachedNode = InReachedNode;
}
```

4.14.3 Member Data Documentation

4.14.3.1 ReachedNode

```
TSharedPtr<FPhraseNode> FParseResult::ReachedNode
```

The Node that was reached in the tree.

Definition at line 79 of file ParseResult.h.

4.14.3.2 Result

```
uint8_t FParseResult::Result
```

The Result of the Propogation.

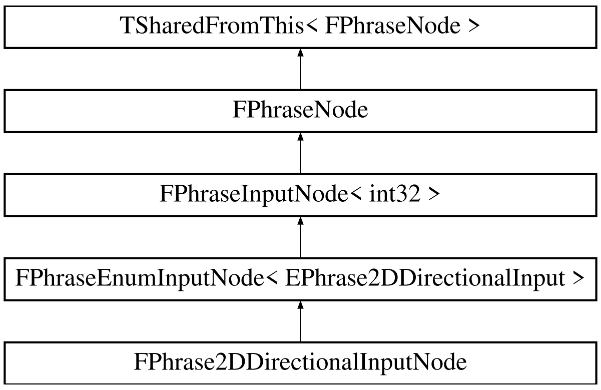
Definition at line 74 of file ParseResult.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/ParseResult.h

4.15 FPhrase2DDirectionalInputNode Class Reference

Inheritance diagram for FPhrase2DDirectionalInputNode:


TSharedFromThis< FPhraseNode >
FPhraseNode
FPhraseInputNode< int32 >
FPhraseEnumInputNode< EPhrase2DDirectionalInput >
FPhrase2DDirectionalInputNode

Public Member Functions

- FPhrase2DDirectionalInputNode (const TCHAR ∗NodeName)
- FPhrase2DDirectionalInputNode (const TCHAR ∗NodeName, TPhraseNodeArray InChildNodes)
- FPhrase2DDirectionalInputNode (const TCHAR ∗NodeName, TDelegate< void(FParseRecord &Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes)
- FPhrase2DDirectionalInputNode (const TCHAR ∗NodeName, TPhraseNodeArray InChildNodes, TDelegate< void(int32 Input)> InOnInputRecieved)
- FPhrase2DDirectionalInputNode (const TCHAR ∗NodeName, TDelegate< void(FParseRecord &Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes, TDelegate< void(int32 Input)> InOnInputRecieved)

Additional Inherited Members

4.15.1 Detailed Description

Definition at line 32 of file PhraseDirectionalInputNode.h.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 FPhrase2DDirectionalInputNode() [1/5]

```
FPhrase2DDirectionalInputNode::FPhrase2DDirectionalInputNode (
            const TCHAR * NodeName )  [inline]
```

Definition at line 35 of file PhraseDirectionalInputNode.h.

```
00036         : FPhraseEnumInputNode<EPhrase2DDirectionalInput>(NodeName)
00037     {}
```

4.15.2.2 FPhrase2DDirectionalInputNode() [2/5]

```
FPhrase2DDirectionalInputNode::FPhrase2DDirectionalInputNode (
            const TCHAR * NodeName,
            TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 39 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhrase2DDirectionalInput>(NodeName, InChildNodes)
    {}
```

4.15.2.3 FPhrase2DDirectionalInputNode() [3/5]

```
FPhrase2DDirectionalInputNode::FPhrase2DDirectionalInputNode (
            const TCHAR * NodeName,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 43 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhrase2DDirectionalInput>(NodeName, InOnPhraseParsed, InChildNodes)
    {}
```

4.15.2.4 FPhrase2DDirectionalInputNode() [4/5]

```
FPhrase2DDirectionalInputNode::FPhrase2DDirectionalInputNode (
            const TCHAR * NodeName,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(int32 Input)> InOnInputRecieved )  [inline]
```

Definition at line 47 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhrase2DDirectionalInput>(NodeName, InChildNodes, InOnInputRecieved)
    {}
```

4.15.2.5 FPhrase2DDirectionalInputNode() [5/5]

```
FPhrase2DDirectionalInputNode::FPhrase2DDirectionalInputNode (
            const TCHAR * NodeName,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(int32 Input)> InOnInputRecieved )  [inline]
```

Definition at line 51 of file PhraseDirectionalInputNode.h.

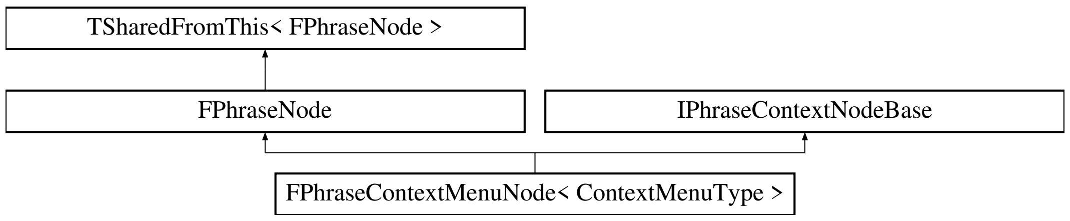
```
        : FPhraseEnumInputNode<EPhrase2DDirectionalInput>(NodeName, InOnPhraseParsed, InChildNodes,
    InOnInputRecieved)
    {}
```

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseDirectionalInputNode.h

4.16 FPhraseContextMenuNode< ContextMenuType > Class Template Reference

Inheritance diagram for FPhraseContextMenuNode< ContextMenuType >:


TSharedFromThis< FPhraseNode >
FPhraseNode
IPhraseContextNodeBase
FPhraseContextMenuNode< ContextMenuType >

Public Member Functions

- FPhraseContextMenuNode (const TCHAR ∗InInputString)
- FPhraseContextMenuNode (const TCHAR ∗InInputString, TPhraseNodeArray InChildNodes)
- FPhraseContextMenuNode (const TCHAR ∗InInputString, TDelegate< TSharedPtr< IMenu >(FParseRecord &Record)> InOnGetMenu, TPhraseNodeArray InChildNodes)
- FPhraseContextMenuNode (const TCHAR ∗InInputString, const float InMenuScalar, TPhraseNodeArray In←ChildNodes)
- FPhraseContextMenuNode (const TCHAR ∗InInputString, const float InMenuScalar, TDelegate< TShared←Ptr< IMenu >(FParseRecord &Record)> InOnGetMenu, TPhraseNodeArray InChildNodes)
- FPhraseContextMenuNode (const TCHAR ∗InInputString, const float InMenuScalar, TDelegate< void(FParseRecord &Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes)
- FPhraseContextMenuNode (const TCHAR ∗InInputString, const float InMenuScalar, TDelegate< TShared←Ptr< IMenu >(FParseRecord &Record)> InOnGetMenu, TDelegate< void(FParseRecord &Record)> In←OnPhraseParsed, TPhraseNodeArray InChildNodes)
- virtual FParseResult ParsePhrase (TArray< FString > &InPhraseWordArray, FParseRecord &InParse←Record) override

  Parses the phrase down the given Node, propagating down child nodes if required.
- virtual FParseResult ParsePhraseAsContext (TArray< FString > &InPhraseWordArray, FParseRecord &In←ParseRecord) override

  Parses the phrase down the given node, propagating down child nodes if required. Missed Pop of the Phrase Array from this Node.

Protected Member Functions

- virtual bool HasContextObject (TArray< UPhraseTreeContextObject ∗ > InContextObjects) const override

  Checks if the Given Context Array Contains Context Objects.
- virtual UPhraseTreeContextObject ∗ CreateContextObject (FParseRecord &Record) override

  Creates a Context Object, using Record Inputs.
- virtual void ConstructContextChildren (TPhraseNodeArray &InChildNodes) override

  Constructs the Context Nodes Children, from Given Child Nodes. Allowing for Inclusion of Utility Nodes in relation to the Context.

Protected Attributes

- const float ContextMenuScalar

  Scalar for the Initialized Menu Elements.
- TDelegate< TSharedPtr< IMenu >(FParseRecord &Record)> OnGetMenu

  Delegate for Initializing of the Menu.

Additional Inherited Members

4.16.1 Detailed Description

template<typename ContextMenuType = UPhraseTreeContextMenuObject>
class FPhraseContextMenuNode< ContextMenuType >

Definition at line 14 of file PhraseContextMenuNode.h.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 FPhraseContextMenuNode() [1/7]

```
template<typename ContextMenuType = UPhraseTreeContextMenuObject>
FPhraseContextMenuNode< ContextMenuType >::FPhraseContextMenuNode (
              const TCHAR * InInputString )  [inline]
```

Definition at line 20 of file PhraseContextMenuNode.h.

```
00021         : FPhraseNode(InInputString)
00022         , ContextMenuScalar(1.0f)
00023     {
00024         this->ChildNodes = TPhraseNodeArray();
00025     };
```

4.16.2.2 FPhraseContextMenuNode() [2/7]

```
template<typename ContextMenuType = UPhraseTreeContextMenuObject>
FPhraseContextMenuNode< ContextMenuType >::FPhraseContextMenuNode (
              const TCHAR * InInputString,
              TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 27 of file PhraseContextMenuNode.h.

```
00028         : FPhraseNode(InInputString)
00029         , ContextMenuScalar(1.0f)
00030     {
00031         ConstructContextChildren(InChildNodes);
00032     };
```

4.16.2.3 FPhraseContextMenuNode() [3/7]

```
template<typename ContextMenuType = UPhraseTreeContextMenuObject>
FPhraseContextMenuNode< ContextMenuType >::FPhraseContextMenuNode (
              const TCHAR * InInputString,
              TDelegate< TSharedPtr< IMenu >(FParseRecord &Record)> InOnGetMenu,
              TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 34 of file PhraseContextMenuNode.h.

```
00035         : FPhraseNode(InInputString)
00036         , ContextMenuScalar(1.0f)
00037         , OnGetMenu(InOnGetMenu)
00038     {
00039         ConstructContextChildren(InChildNodes);
00040     };
```

4.16.2.4 FPhraseContextMenuNode() [4/7]

```
template<typename ContextMenuType = UPhraseTreeContextMenuObject>
FPhraseContextMenuNode< ContextMenuType >::FPhraseContextMenuNode (
              const TCHAR * InInputString,
              const float InMenuScalar,
              TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 42 of file PhraseContextMenuNode.h.

```
        : FPhraseNode(InInputString)
        , ContextMenuScalar(InMenuScalar)
    {
        ConstructContextChildren(InChildNodes);
    };
```

4.16.2.5 FPhraseContextMenuNode() [5/7]

```
template<typename ContextMenuType = UPhraseTreeContextMenuObject>
FPhraseContextMenuNode< ContextMenuType >::FPhraseContextMenuNode (
              const TCHAR * InInputString,
              const float InMenuScalar,
              TDelegate< TSharedPtr< IMenu >(FParseRecord &Record)> InOnGetMenu,
              TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 49 of file PhraseContextMenuNode.h.

```
        : FPhraseNode(InInputString)
        , ContextMenuScalar(InMenuScalar)
        , OnGetMenu(InOnGetMenu)
    {
        ConstructContextChildren(InChildNodes);
    }
```

4.16.2.6 FPhraseContextMenuNode() [6/7]

```
template<typename ContextMenuType = UPhraseTreeContextMenuObject>
FPhraseContextMenuNode< ContextMenuType >::FPhraseContextMenuNode (
              const TCHAR * InInputString,
              const float InMenuScalar,
              TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
              TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 57 of file PhraseContextMenuNode.h.

```
        : FPhraseNode(InInputString, InOnPhraseParsed)
        , ContextMenuScalar(InMenuScalar)
    {
        ConstructContextChildren(InChildNodes);
    }
```

4.16.2.7 FPhraseContextMenuNode() [7/7]

```
template<typename ContextMenuType = UPhraseTreeContextMenuObject>
FPhraseContextMenuNode< ContextMenuType >::FPhraseContextMenuNode (
              const TCHAR * InInputString,
              const float InMenuScalar,
              TDelegate< TSharedPtr< IMenu >(FParseRecord &Record)> InOnGetMenu,
              TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
              TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 64 of file PhraseContextMenuNode.h.

```
        : FPhraseNode(InInputString, InOnPhraseParsed)
        , ContextMenuScalar(InMenuScalar)
        , OnGetMenu(InOnGetMenu)
    {
        ConstructContextChildren(InChildNodes);
    }
```

4.16.2.8 ∼FPhraseContextMenuNode()

```
template<typename ContextMenuType = UPhraseTreeContextMenuObject>
FPhraseContextMenuNode< ContextMenuType >::∼FPhraseContextMenuNode ( )  [inline]
```

Definition at line 72 of file PhraseContextMenuNode.h.

```
    {

    }
```

4.16.3 Member Function Documentation

4.16.3.1 ConstructContextChildren()

```
template<typename ContextMenuType >
void FPhraseContextMenuNode< ContextMenuType >::ConstructContextChildren (
              TPhraseNodeArray & InChildNodes )  [override], [protected], [virtual]
```

Constructs the Context Nodes Children, from Given Child Nodes. Allowing for Inclusion of Utility Nodes in relation to the Context.

Parameters

InChildNodes	- An Array of the Nodes Children.

Definition at line 225 of file PhraseContextMenuNode.h.

```
{
    // Construct Context Specific Children Nodes,
    // With Linked Functionality to the Context Menu Object and Root Node.
    TSharedPtr<FPhraseEventNode> CloseContextNode = MakeShared<FPhraseEventNode>();
    CloseContextNode->OnPhraseParsed.BindLambda(
        [this](FParseRecord& Record) {

            UPhraseTreeContextMenuObject* ContextMenu = Record.GetContextObj<UPhraseTreeContextMenuObject>();
            if (ContextMenu->GetContextRoot() == this->AsShared())
            {
                ContextMenu->Close();
                ContextMenu->RemoveFromRoot();

                Record.PopContextObj();
            }
        }
    );

    this->ChildNodes = TPhraseNodeArray{
        MakeShared<FPhraseNode>(TEXT("CLOSE"),
        TPhraseNodeArray {
            CloseContextNode
        })
    };

    this->ChildNodes.Append(InChildNodes);
}
```

4.16.3.2 CreateContextObject()

template<typename ContextMenuType >
UPhraseTreeContextObject ∗ FPhraseContextMenuNode< ContextMenuType >::CreateContextObject (
FParseRecord & *Record*) [override], [protected], [virtual]

Creates a Context Object, using Record Inputs.

Returns

The Created Context Object, otherwise nullptr

Implements IPhraseContextNodeBase.

Definition at line 200 of file PhraseContextMenuNode.h.

```
{
    if (!OnGetMenu.IsBound())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("OnGetMenu Delegate Not Bound. Cannot Create Context Object, linked to a Menu."));
        return nullptr;
    }

    TSharedPtr<IMenu> NewMenu = OnGetMenu.Execute(Record);

    if (!NewMenu.IsValid())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("OnGetMenu Delegate Returned Invalid Menu. Cannot Create Context Object."));
        return nullptr;
    }

    ContextMenuType* NewContextObject = NewObject<ContextMenuType>();
    NewContextObject->Init(NewMenu.ToSharedRef(), this->AsShared());

    NewContextObject->ScaleMenu(ContextMenuScalar);

    return NewContextObject;
}
```

4.16.3.3 HasContextObject()

template<typename ContextMenuType >
bool FPhraseContextMenuNode< ContextMenuType >::HasContextObject (
TArray< UPhraseTreeContextObject ∗ > *InContextObjects*) const [override], [protected], [virtual]

Checks if the Given Context Array Contains Context Objects.

Parameters

InContextObjects	- The Array To Check For Context Objects.

Returns

True, if their is Context Objects in the Given Array.

Implements IPhraseContextNodeBase.

Definition at line 186 of file PhraseContextMenuNode.h.

```
{
    for (auto& ContextObject : InContextObjects)
    {
        if (ContextObject->IsA(ContextMenuType::StaticClass()) && ContextObject->GetContextRoot() ==
AsShared())
        {
            return true;
        }
    }

    return false;
}
```

4.16.3.4 ParsePhrase()

```
template<typename ContextMenuType >
FParseResult FPhraseContextMenuNode< ContextMenuType >::ParsePhrase (
            TArray< FString > & InPhraseWordArray,
            FParseRecord & InParseRecord )  [override], [virtual]
```

Parses the phrase down the given Node, propagating down child nodes if required.

Parameters

InPhraseWordArray	The Array of Phrase Strings to Propogate against.
InParseRecord	The Record of Propagation of collected context's and inputs.

Returns

Returns the Result of the propogation, including any key nodes met.

Reimplemented from FPhraseNode.

Definition at line 138 of file PhraseContextMenuNode.h.

```
{
    if (!HasContextObject(InParseRecord.GetContextStack()))
    {
        UPhraseTreeContextObject* NewObject = CreateContextObject(InParseRecord);

        InParseRecord.PushContextObj(NewObject);
    }

    if (InPhraseWordArray.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Emptied Phrase Array ||"))

        return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
    }

    InPhraseWordArray.Pop();

    OnPhraseParsed.ExecuteIfBound(InParseRecord);

    return ParseChildren(InPhraseWordArray, InParseRecord);

    return FPhraseNode::ParsePhrase(InPhraseWordArray, InParseRecord);
}
```

4.16.3.5 ParsePhraseAsContext()

```
template<typename ContextMenuType >
FParseResult FPhraseContextMenuNode< ContextMenuType >::ParsePhraseAsContext (
            TArray< FString > & InPhraseWordArray,
            FParseRecord & InParseRecord )  [inline], [override], [virtual]
```

Parses the phrase down the given node, propagating down child nodes if required. Missed Pop of the Phrase Array from this Node.

Parameters

InPhraseWordArray	
InParseRecord	

Returns

Returns the Result of the propogation, including any key nodes met.

Reimplemented from FPhraseNode.

Definition at line 164 of file PhraseContextMenuNode.h.

```
{
    if (!HasContextObject(InParseRecord.GetContextStack()))
    {
        UPhraseTreeContextObject* NewObject = CreateContextObject(InParseRecord);

        InParseRecord.PushContextObj(NewObject);
    }

    if (InPhraseWordArray.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Emptied Phrase Array ||"))

        return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
    }

    OnPhraseParsed.ExecuteIfBound(InParseRecord);

    return ParseChildren(InPhraseWordArray, InParseRecord);
}
```

4.16.4 Member Data Documentation

4.16.4.1 ContextMenuScalar

```
template<typename ContextMenuType = UPhraseTreeContextMenuObject>
const float FPhraseContextMenuNode< ContextMenuType >::ContextMenuScalar  [protected]
```

Scalar for the Initialized Menu Elements.

Definition at line 129 of file PhraseContextMenuNode.h.

4.16.4.2 OnGetMenu

```
template<typename ContextMenuType = UPhraseTreeContextMenuObject>
TDelegate<TSharedPtr<IMenu>(FParseRecord& Record)> FPhraseContextMenuNode< ContextMenuType
>::OnGetMenu  [protected]
```

Delegate for Initializing of the Menu.

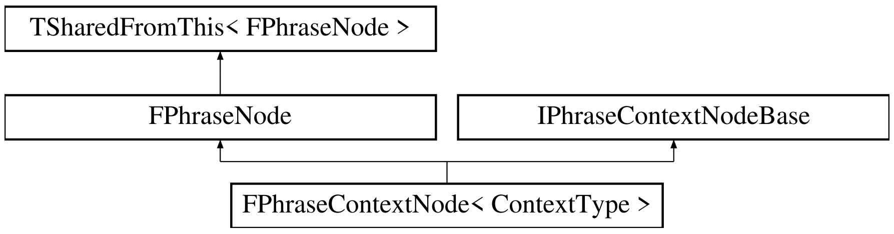
Definition at line 134 of file PhraseContextMenuNode.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseContextMenuNode.h

4.17 FPhraseContextNode< ContextType > Class Template Reference

Inheritance diagram for FPhraseContextNode< ContextType >:


TSharedFromThis< FPhraseNode >
FPhraseNode
IPhraseContextNodeBase
FPhraseContextNode< ContextType >

Public Member Functions

- FPhraseContextNode (const TCHAR ∗InInputString)
- FPhraseContextNode (const TCHAR ∗InInputString, TPhraseNodeArray InChildNodes)
- FPhraseContextNode (const TCHAR ∗InInputString, TDelegate< void(FParseRecord &Record)> InOn↩ PhraseParsed, TPhraseNodeArray InChildNodes)
- virtual FParseResult ParsePhrase (TArray< FString > &InPhraseWordArray, FParseRecord &InParse↩ Record) override

  Parses The Phrase Down This Node, Propagating Down Any Child Nodes If Required.
- virtual FParseResult ParsePhraseAsContext (TArray< FString > &InPhraseWordArray, FParseRecord &In↩ ParseRecord) override

  Parses the Phrase Down This Node, Propagating Down Any Child Nodes If Required. Does not Pop the Phrase Array.

Protected Member Functions

- bool HasContextObject (TArray< UPhraseTreeContextObject ∗ > InContextObjects) const

  Checks if the Given Context Array Contains Context Objects.
- virtual UPhraseTreeContextObject ∗ CreateContextObject (FParseRecord &Record)

  Creates a Context Object, using Record Inputs.
- virtual void ConstructContextChildren (TPhraseNodeArray &InChildNodes)

Additional Inherited Members

4.17.1 Detailed Description

template<class ContextType = UPhraseTreeContextObject>
class FPhraseContextNode< ContextType >

Definition at line 14 of file PhraseContextNode.h.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 FPhraseContextNode() [1/3]

```
template<class ContextType = UPhraseTreeContextObject>
FPhraseContextNode< ContextType >::FPhraseContextNode (
              const TCHAR * InInputString )  [inline]
```

Definition at line 18 of file PhraseContextNode.h.

```
        : FPhraseNode(InInputString)
    {
        static_assert(std::is_base_of<UPhraseTreeContextObject, ContextType>::value, "ContextType must
      be a subclass of UPhraseTreeContextObject");

        TPhraseNodeArray EmptyArray = TPhraseNodeArray();
        ConstructContextChildren(EmptyArray);
    }
```

4.17.2.2 FPhraseContextNode() [2/3]

```
template<class ContextType = UPhraseTreeContextObject>
FPhraseContextNode< ContextType >::FPhraseContextNode (
              const TCHAR * InInputString,
              TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 27 of file PhraseContextNode.h.

```
        : FPhraseNode(InInputString, InChildNodes)
    {
        static_assert(std::is_base_of<UPhraseTreeContextObject, ContextType>::value, "ContextType must
      be a subclass of UPhraseTreeContextObject");

        ConstructContextChildren(InChildNodes);
    }
```

4.17.2.3 FPhraseContextNode() [3/3]

```
template<class ContextType = UPhraseTreeContextObject>
FPhraseContextNode< ContextType >::FPhraseContextNode (
            const TCHAR * InInputString,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 35 of file PhraseContextNode.h.

```
        : FPhraseNode(InInputString, InOnPhraseParsed)
    {
        static_assert(std::is_base_of<UPhraseTreeContextObject, ContextType>::value, "ContextType must
be a subclass of UPhraseTreeContextObject");

        ConstructContextChildren(InChildNodes);
    }
```

4.17.2.4 ∼FPhraseContextNode()

```
template<class ContextType = UPhraseTreeContextObject>
FPhraseContextNode< ContextType >::∼FPhraseContextNode ( )  [inline]
```

Definition at line 43 of file PhraseContextNode.h.

```
    {

    }
```

4.17.3 Member Function Documentation

4.17.3.1 ConstructContextChildren()

```
template<class ContextType >
void FPhraseContextNode< ContextType >::ConstructContextChildren (
            TPhraseNodeArray & InChildNodes )  [protected], [virtual]
```

Definition at line 132 of file PhraseContextNode.h.

```
{
    TSharedPtr<FPhraseEventNode> CloseContextNode = MakeShared<FPhraseEventNode>();
    CloseContextNode->OnPhraseParsed.BindLambda(
        [this](FParseRecord& Record) {

            UPhraseTreeContextObject* ContextObject = Record.GetContextObj();
            if (ContextObject->GetContextRoot() == this->AsShared())
            {
                ContextObject->Close();
                ContextObject->RemoveFromRoot();

                Record.PopContextObj();
            }
        }
    );

    this->ChildNodes = TPhraseNodeArray{
        MakeShared<FPhraseNode>(TEXT("CLOSE"),
        TPhraseNodeArray {
            CloseContextNode
        })
    };

    this->ChildNodes.Append(InChildNodes);
}
```

4.17.3.2 CreateContextObject()

```
template<class ContextType >
UPhraseTreeContextObject ∗ FPhraseContextNode< ContextType >::CreateContextObject (
            FParseRecord & Record ) [protected], [virtual]
```

Creates a Context Object, using Record Inputs.

Returns

The Created Context Object, otherwise nullptr

Implements IPhraseContextNodeBase.

Definition at line 122 of file PhraseContextNode.h.

```
{
    ContextType* NewContextObject = NewObject<ContextType>();
    NewContextObject->Init();
    NewContextObject->SetContextRootNode(AsShared());

    return NewContextObject;
}
```

4.17.3.3 HasContextObject()

```
template<class ContextType >
bool FPhraseContextNode< ContextType >::HasContextObject (
            TArray< UPhraseTreeContextObject ∗ > InContextObjects ) const [protected], [virtual]
```

Checks if the Given Context Array Contains Context Objects.

Parameters

InContextObjects	- The Array To Check For Context Objects.

Returns

True, if their is Context Objects in the Given Array.

Implements IPhraseContextNodeBase.

Definition at line 107 of file PhraseContextNode.h.

```
{
    for (auto& ContextObject : InContextObjects)
    {
        if (ContextObject->IsA(ContextType::StaticClass()) && ContextObject->GetContextRoot() ==
 AsShared())
        {
            return true;
        }
    }

    return false;
}
```

4.17.3.4 ParsePhrase()

```
template<class ContextType >
FParseResult FPhraseContextNode< ContextType >::ParsePhrase (
            TArray< FString > & InPhraseWordArray,
            FParseRecord & InParseRecord )  [override], [virtual]
```

Parses The Phrase Down This Node, Propagating Down Any Child Nodes If Required.

Parameters

InPhraseWordArray	- The Current Array of Transcription Phrases.
InParseRecord	- The Parse Record of the Current Propagation.

Returns

The Result of the Parsing of the Phrase, and any Propagation.

Reimplemented from FPhraseNode.

Definition at line 71 of file PhraseContextNode.h.

```
{
    if (!HasContextObject(InParseRecord.GetContextStack()))
    {
        UPhraseTreeContextObject* NewObject = CreateContextObject(InParseRecord);

        InParseRecord.PushContextObj(NewObject);
    }

    return FPhraseNode::ParsePhrase(InPhraseWordArray, InParseRecord);
}
```

4.17.3.5 ParsePhraseAsContext()

```
template<class ContextType >
FParseResult FPhraseContextNode< ContextType >::ParsePhraseAsContext (
            TArray< FString > & InPhraseWordArray,
            FParseRecord & InParseRecord )  [override], [virtual]
```

Parses the Phrase Down This Node, Propagating Down Any Child Nodes If Required. Does not Pop the Phrase Array.

Parameters

InPhraseWordArray	- The Current Array of Transcription Phrases.
InParseRecord	- The Parse Record of the Current Propagation.

Returns

The Result of the Parsing of the Phrase, and any Propagation.

Reimplemented from FPhraseNode.

Definition at line 84 of file PhraseContextNode.h.

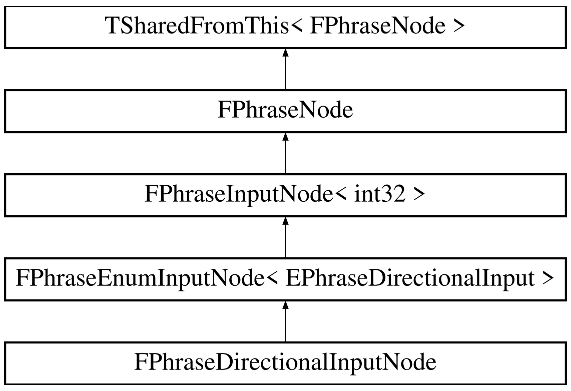
```
{
    if (!HasContextObject(InParseRecord.GetContextStack()))
    {
        UPhraseTreeContextObject* NewObject = CreateContextObject(InParseRecord);

        InParseRecord.PushContextObj(NewObject);
    }

    if (InPhraseWordArray.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Emptied Phrase Array ||"))

            return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
    }

    OnPhraseParsed.ExecuteIfBound(InParseRecord);

    // Pass
    return ParseChildren(InPhraseWordArray, InParseRecord);
}
```

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseContextNode.h

4.18 FPhraseDirectionalInputNode Class Reference

Inheritance diagram for FPhraseDirectionalInputNode:


TSharedFromThis< FPhraseNode >
FPhraseNode
FPhraseInputNode< int32 >
FPhraseEnumInputNode< EPhraseDirectionalInput >
FPhraseDirectionalInputNode

Public Member Functions

- FPhraseDirectionalInputNode (const TCHAR ∗NodeName)
- FPhraseDirectionalInputNode (const TCHAR ∗NodeName, TPhraseNodeArray InChildNodes)
- FPhraseDirectionalInputNode (const TCHAR ∗NodeName, TDelegate< void(FParseRecord &Record)> In↩ OnPhraseParsed, TPhraseNodeArray InChildNodes)
- FPhraseDirectionalInputNode (const TCHAR ∗NodeName, TPhraseNodeArray InChildNodes, TDelegate< void(int32 Input)> InOnInputRecieved)
- FPhraseDirectionalInputNode (const TCHAR ∗NodeName, TDelegate< void(FParseRecord &Record)> In↩ OnPhraseParsed, TPhraseNodeArray InChildNodes, TDelegate< void(int32 Input)> InOnInputRecieved)

Additional Inherited Members

4.18.1 Detailed Description

Definition at line 8 of file PhraseDirectionalInputNode.h.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 FPhraseDirectionalInputNode() [1/5]

```
FPhraseDirectionalInputNode::FPhraseDirectionalInputNode (
            const TCHAR * NodeName )  [inline]
```

Definition at line 11 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhraseDirectionalInput>(NodeName)
    {}
```

4.18.2.2 FPhraseDirectionalInputNode() [2/5]

```
FPhraseDirectionalInputNode::FPhraseDirectionalInputNode (
            const TCHAR * NodeName,
            TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 15 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhraseDirectionalInput>(NodeName, InChildNodes)
    {}
```

4.18.2.3 FPhraseDirectionalInputNode() [3/5]

```
FPhraseDirectionalInputNode::FPhraseDirectionalInputNode (
            const TCHAR * NodeName,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 19 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhraseDirectionalInput>(NodeName, InOnPhraseParsed, InChildNodes)
    {}
```

4.18.2.4 FPhraseDirectionalInputNode() [4/5]

```
FPhraseDirectionalInputNode::FPhraseDirectionalInputNode (
            const TCHAR * NodeName,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(int32 Input)> InOnInputRecieved )  [inline]
```

Definition at line 23 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhraseDirectionalInput>(NodeName, InChildNodes, InOnInputRecieved)
    {}
```

4.18.2.5 FPhraseDirectionalInputNode() [5/5]

```
FPhraseDirectionalInputNode::FPhraseDirectionalInputNode (
            const TCHAR * NodeName,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(int32 Input)> InOnInputRecieved ) [inline]
```

Definition at line 27 of file PhraseDirectionalInputNode.h.

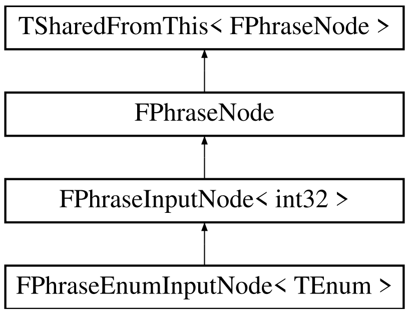
```
00028         : FPhraseEnumInputNode<EPhraseDirectionalInput>(NodeName, InOnPhraseParsed, InChildNodes,
      InOnInputRecieved)
00029     {}
```

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseDirectionalInputNode.h

4.19 FPhraseEnumInputNode< TEnum > Class Template Reference

Inheritance diagram for FPhraseEnumInputNode< TEnum >:


TSharedFromThis< FPhraseNode >
FPhraseNode
FPhraseInputNode< int32 >
FPhraseEnumInputNode< TEnum >

Public Member Functions

- FPhraseEnumInputNode (const TCHAR ∗InInputString)
- FPhraseEnumInputNode (const TCHAR ∗InInputString, TPhraseNodeArray InChildNodes)
- FPhraseEnumInputNode (const TCHAR ∗InInputString, TDelegate< void(FParseRecord &Record)> InOn↩ PhraseParsed, TPhraseNodeArray InChildNodes)
- FPhraseEnumInputNode (const TCHAR ∗InInputString, TPhraseNodeArray InChildNodes, TDelegate< void(int32 Input)> InOnInputRecieved)
- FPhraseEnumInputNode (const TCHAR ∗InInputString, TDelegate< void(FParseRecord &Record)> InOn↩ PhraseParsed, TPhraseNodeArray InChildNodes, TDelegate< void(int32 Input)> InOnInputRecieved)

Protected Member Functions

- virtual bool MeetsInputRequirements (const FString &InPhrase) override

  Checks if the Given Phrase Meets Requirements for usage as Input. In Correlation to this Nodes Input Specifications.
- virtual bool RecordInput (const FString &InInput, FParseRecord &OutParseRecord) override

  Records the Input onto the Parse Record.

Additional Inherited Members

4.19.1 Detailed Description

template<typename TEnum>
class FPhraseEnumInputNode< TEnum >

Definition at line 13 of file PhraseEnumInputNode.h.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 FPhraseEnumInputNode() [1/5]

```
template<typename TEnum >
FPhraseEnumInputNode< TEnum >::FPhraseEnumInputNode (
            const TCHAR * InInputString )
```

Definition at line 9 of file PhraseEnumInputNode.cpp.

```
    : FPhraseInputNode(NodeName)
{
    static_assert(TIsEnum<TEnum>::Value, "Passed EnumType Must be an Enum.");
};
```

4.19.2.2 FPhraseEnumInputNode() [2/5]

```
template<typename TEnum >
FPhraseEnumInputNode< TEnum >::FPhraseEnumInputNode (
            const TCHAR * InInputString,
            TPhraseNodeArray InChildNodes )
```

Definition at line 16 of file PhraseEnumInputNode.cpp.

```
    : FPhraseInputNode(NodeName, InChildNodes)
{
    static_assert(TIsEnum<TEnum>::Value, "Passed EnumType Must be an Enum");
}
```

4.19.2.3 FPhraseEnumInputNode() [3/5]

```
template<typename TEnum >
FPhraseEnumInputNode< TEnum >::FPhraseEnumInputNode (
            const TCHAR * InInputString,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes )
```

Definition at line 23 of file PhraseEnumInputNode.cpp.

```
    : FPhraseInputNode(InInputString, InOnPhraseParsed, InChildNodes)
{
    static_assert(TIsEnum<TEnum>::Value, "Passed EnumType Must be an Enum");
}
```

4.19.2.4 FPhraseEnumInputNode() [4/5]

```
template<typename TEnum >
FPhraseEnumInputNode< TEnum >::FPhraseEnumInputNode (
            const TCHAR * InInputString,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(int32 Input)> InOnInputRecieved )
```

Definition at line 30 of file PhraseEnumInputNode.cpp.

```
    : FPhraseInputNode(InInputString, InChildNodes, InOnInputRecieved)
{
    static_assert(TIsEnum<TEnum>::Value, "Passed EnumType Must be an Enum");
}
```

4.19.2.5 FPhraseEnumInputNode() [5/5]

```
template<typename TEnum >
FPhraseEnumInputNode< TEnum >::FPhraseEnumInputNode (
            const TCHAR * InInputString,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(int32 Input)> InOnInputRecieved )
```

Definition at line 37 of file PhraseEnumInputNode.cpp.

```
    : FPhraseInputNode(InInputString, InOnPhraseParsed, InChildNodes, InOnInputRecieved)
{
    static_assert(TIsEnum<TEnum>::Value, "Passed EnumType Must be an Enum");
}
```

4.19.2.6 ∼FPhraseEnumInputNode()

```
template<typename TEnum >
FPhraseEnumInputNode< TEnum >::∼FPhraseEnumInputNode
```

Definition at line 44 of file PhraseEnumInputNode.cpp.

```
{

}
```

4.19.3 Member Function Documentation

4.19.3.1 MeetsInputRequirements()

```
template<typename TEnum >
bool FPhraseEnumInputNode< TEnum >::MeetsInputRequirements (
            const FString & InPhrase )  [override], [protected], [virtual]
```

Checks if the Given Phrase Meets Requirements for usage as Input. In Correlation to this Nodes Input Specifications.

Parameters

InPhrase	- The Phrase To Check If It Meets Requirements.

Returns

True, if the Phrase Meets Requirements. Otherwise False.

Reimplemented from FPhraseInputNode< int32 >.

Definition at line 50 of file PhraseEnumInputNode.cpp.

```
{
    UEnum* EnumPtr = StaticEnum<TEnum>();
    if (!EnumPtr)
    {
        UE_LOG(LogTemp, Error, TEXT("FPhraseEnumInputNode::MeetsInputRequirements: EnumPtr is NULL"));
        return false;
    }

    return EnumPtr->IsValidEnumName(*EnumPtr->GenerateFullEnumName(*InPhrase.ToUpper()));
}
```

4.19.3.2 RecordInput()

```
template<typename TEnum >
bool FPhraseEnumInputNode< TEnum >::RecordInput (
            const FString & InInput,
            FParseRecord & OutParseRecord )  [override], [protected], [virtual]
```

Records the Input onto the Parse Record.

Parameters

InInput	- The Phrase To Record onto the Parse Record.
OutParseRecord	- Returns the Updated ParseRecord.

Returns

True, if the Input Was Successful in Recording. Otherwise False.

Reimplemented from FPhraseInputNode< int32 >.

Definition at line 63 of file PhraseEnumInputNode.cpp.

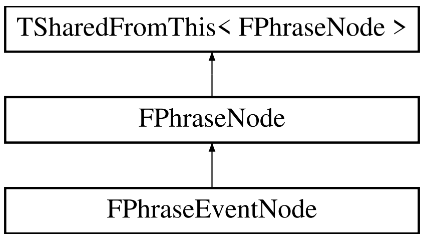
```
{
    UEnum* EnumPtr = StaticEnum<TEnum>();
    if (!EnumPtr)
    {
        UE_LOG(LogTemp, Error, TEXT("FPhraseEnumInputNode::RecordInput: EnumPtr is NULL"));
        return false;
    }

    int32 Val = EnumPtr->GetValueByNameString(EnumPtr->GenerateFullEnumName(*InInput.ToUpper()));
    if (Val == INDEX_NONE)
    {
        return false;
    }

    UParseEnumInput* ParseInput = MakeParseInput<UParseEnumInput>();
    ParseInput->SetValue(Val);
    ParseInput->SetEnumType(EnumPtr);

    OutParseRecord.AddPhraseInput(BoundPhrase, ParseInput);

    return true;
}
```

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseEnumInputNode.h
- Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseEnumInputNode.cpp

4.20 FPhraseEventNode Class Reference

Inheritance diagram for FPhraseEventNode:



Public Member Functions

- FPhraseEventNode (TDelegate< void(FParseRecord &)> InEvent)
- FPhraseEventNode (TFunction< void(FParseRecord &)> InEventFunction)
- virtual bool IsLeafNode () const override

  Checks if the Node is a Leaf Node.
- virtual bool RequiresPhrase (const FString InPhrase) override

  Checks if the Node Requires the Given Phrase.
- virtual bool RequiresPhrase (const FString InPhrase, int32 &OutDistance)

  Checks if the Node Requires the Given Phrase, and Returns the Distance of the Phrase.
- virtual FParseResult ParsePhrase (TArray< FString > &InPhraseArray, FParseRecord &InParseRecord) override

  Parses The Phrase Down This Node, Propagating Down Any Child Nodes If Required.

Additional Inherited Members

4.20.1 Detailed Description

Definition at line 11 of file PhraseEventNode.h.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 FPhraseEventNode() [1/3]

```
FPhraseEventNode::FPhraseEventNode ( )
```

Definition at line 7 of file PhraseEventNode.cpp.

```
    : FPhraseNode(TEXT("EVENT_NODE"))
{
    OnPhraseParsed = TDelegate<void(FParseRecord&)>();
}
```

4.20.2.2 FPhraseEventNode() [2/3]

```
FPhraseEventNode::FPhraseEventNode (
            TDelegate< void(FParseRecord &)> InEvent )
```

Definition at line 13 of file PhraseEventNode.cpp.

```
    : FPhraseNode(TEXT("EVENT_NODE"), InEvent)
{

}
```

4.20.2.3 FPhraseEventNode() [3/3]

```
FPhraseEventNode::FPhraseEventNode (
            TFunction< void(FParseRecord &)> InEventFunction )
```

Definition at line 19 of file PhraseEventNode.cpp.

```
    : FPhraseNode(TEXT("EVENT_NODE"), TDelegate<void(FParseRecord&)>::CreateLambda(InEventFunction))
{

}
```

4.20.2.4 ∼FPhraseEventNode()

```
FPhraseEventNode::∼FPhraseEventNode ( )
```

Definition at line 25 of file PhraseEventNode.cpp.

```
{

}
```

4.20.3 Member Function Documentation

4.20.3.1 IsLeafNode()

```
virtual bool FPhraseEventNode::IsLeafNode ( ) const  [inline], [override], [virtual]
```

Checks if the Node is a Leaf Node.

Returns

True, if the Node is a Leaf Node. Otherwise False.

Reimplemented from FPhraseNode.

Definition at line 21 of file PhraseEventNode.h.

```
{ return true; }
```

4.20.3.2 ParsePhrase()

```
FParseResult FPhraseEventNode::ParsePhrase (
            TArray< FString > & InPhraseWordArray,
            FParseRecord & InParseRecord )  [override], [virtual]
```

Parses The Phrase Down This Node, Propagating Down Any Child Nodes If Required.

Parameters

InPhraseWordArray	- The Current Array of Transcription Phrases.
InParseRecord	- The Parse Record of the Current Propagation.

Returns

The Result of the Parsing of the Phrase, and any Propagation.

Reimplemented from FPhraseNode.

Definition at line 41 of file PhraseEventNode.cpp.

```
{
    if (OnPhraseParsed.ExecuteIfBound(InParseRecord))
    {
        return FParseResult(PHRASE_PARSED_AND_EXECUTED);
    }

    UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Unable to Execute Event ||"))

    return FParseResult(PHRASE_UNABLE_TO_PARSE, AsShared());
}
```

4.20.3.3 RequiresPhrase() [1/2]

```
bool FPhraseEventNode::RequiresPhrase (
            const FString InPhrase )  [override], [virtual]
```

Checks if the Node Requires the Given Phrase.

Parameters

InPhrase	- The Phrase To Check if Required By The Node.

Returns

True, if the Phrase is Required. Otherwise False.

Reimplemented from FPhraseNode.

Definition at line 30 of file PhraseEventNode.cpp.
```
00031 {
00032     return true;
00033 }
```

4.20.3.4 RequiresPhrase() [2/2]

```
bool FPhraseEventNode::RequiresPhrase (
            const FString InPhrase,
            int32 & OutDistance ) [virtual]
```

Checks if the Node Requires the Given Phrase, and Returns the Distance of the Phrase.

Parameters

InPhrase	- The Phrase To Check if Required By The Node.
OutDistance	- The Returned Distancing from the Target Phrase To The BoundPhrase.

Returns

True, if the Phrase is Required. Otherwise False.

Reimplemented from FPhraseNode.

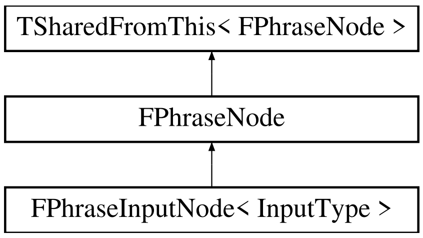
Definition at line 35 of file PhraseEventNode.cpp.
```
00036 {
00037     OutDistance = 0;
00038     return true;
00039 }
```

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseEventNode.h
- Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseEventNode.cpp

4.21 FPhraseInputNode< InputType > Class Template Reference

Inheritance diagram for FPhraseInputNode< InputType >:



Public Member Functions

- FPhraseInputNode (const TCHAR ∗InInputString)
- FPhraseInputNode (const TCHAR ∗InInputString, TPhraseNodeArray InChildNodes)
- FPhraseInputNode (const TCHAR ∗InInputString, TDelegate< void(FParseRecord &Record)> InOn←PhraseParsed, TPhraseNodeArray InChildNodes)
- FPhraseInputNode (const TCHAR ∗InInputString, TPhraseNodeArray InChildNodes, TDelegate< void(InputType Input)> InOnInputRecieved)
- FPhraseInputNode (const TCHAR ∗InInputString, TDelegate< void(FParseRecord &Record)> InOn←PhraseParsed, TPhraseNodeArray InChildNodes, TDelegate< void(InputType Input)> InOnInputRecieved)
- virtual bool RequiresPhrase (const FString InPhrase) override

    Checks if the Node Requires the Given Phrase.
- virtual bool RequiresPhrase (const FString InPhrase, int32 &OutDistance) override

    Checks if the Node Requires the Given Phrase, and Returns the Distance of the Phrase.
- virtual FParseResult ParsePhrase (TArray< FString > &InPhraseArray, FParseRecord &InParseRecord) override

    Parses The Phrase Down This Node, Propagating Down Any Child Nodes If Required.

Public Attributes

- TDelegate< void(InputType ReceivedInput)> OnInputReceived

Protected Member Functions

- virtual bool MeetsInputRequirements (const FString &InPhrase)

    Checks if the Given Phrase Meets Requirements for usage as Input. In Correlation to this Nodes Input Specifications.
- virtual bool RecordInput (const FString &InInput, FParseRecord &OutParseRecord)

    Records the Input onto the Parse Record.

Additional Inherited Members

4.21.1 Detailed Description

template<typename InputType = int32>
class FPhraseInputNode< InputType >

Definition at line 12 of file PhraseInputNode.h.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 FPhraseInputNode() [1/5]

```
template<typename InputType >
FPhraseInputNode< InputType >::FPhraseInputNode (
            const TCHAR * InInputString )
```

Definition at line 10 of file PhraseInputNode.cpp.

```
    : FPhraseNode(InInputString)
{

}
```

4.21.2.2 FPhraseInputNode() [2/5]

```
template<typename InputType >
FPhraseInputNode< InputType >::FPhraseInputNode (
            const TCHAR * InInputString,
            TPhraseNodeArray InChildNodes )
```

Definition at line 17 of file PhraseInputNode.cpp.

```
    : FPhraseNode(InInputString, InChildNodes)
{

}
```

4.21.2.3 FPhraseInputNode() [3/5]

```
template<typename InputType >
FPhraseInputNode< InputType >::FPhraseInputNode (
            const TCHAR * InInputString,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes )
```

Definition at line 24 of file PhraseInputNode.cpp.

```
    : FPhraseNode(InInputString, InOnPhraseParsed, InChildNodes)
{

}
```

4.21.2.4 FPhraseInputNode() [4/5]

```
template<typename InputType >
FPhraseInputNode< InputType >::FPhraseInputNode (
            const TCHAR * InInputString,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(InputType Input)> InOnInputRecieved )
```

Definition at line 31 of file PhraseInputNode.cpp.

```
    : FPhraseNode(InInputString, InChildNodes)
{
    OnInputReceived = InOnInputRecieved;
}
```

4.21.2.5 FPhraseInputNode() [5/5]

```
template<typename InputType >
FPhraseInputNode< InputType >::FPhraseInputNode (
            const TCHAR * InInputString,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(InputType Input)> InOnInputRecieved )
```

Definition at line 38 of file PhraseInputNode.cpp.

```
    : FPhraseNode(InInputString, InOnPhraseParsed, InChildNodes)
{
    OnInputReceived = InOnInputRecieved;
}
```

4.21.2.6 ∼FPhraseInputNode()

```
template<typename InputType >
FPhraseInputNode< InputType >::∼FPhraseInputNode
```

Definition at line 45 of file PhraseInputNode.cpp.

```
{

}
```

4.21.3 Member Function Documentation

4.21.3.1 MeetsInputRequirements()

```
template<typename InputType >
bool FPhraseInputNode< InputType >::MeetsInputRequirements (
            const FString & InPhrase ) [protected], [virtual]
```

Checks if the Given Phrase Meets Requirements for usage as Input. In Correlation to this Nodes Input Specifications.

Parameters

InPhrase	- The Phrase To Check If It Meets Requirements.

Returns

True, if the Phrase Meets Requirements. Otherwise False.

Reimplemented in FPhraseEnumInputNode< TEnum >, FPhraseEnumInputNode< EPhrase2DDirectionalInput >, FPhraseEnumInputNode< EPhraseDirectionalInput >, FPhraseEnumInputNode< EPhrasePositionalInput >, FPhraseEnumInputNode< EPhraseScrollInput >, and FPhraseStringInputNode.

Definition at line 104 of file PhraseInputNode.cpp.

```
{
    return InPhrase.IsNumeric() || NumericParser::IsValidNumeric(InPhrase, false);
}
```

4.21.3.2 ParsePhrase()

```
template<typename InputType >
FParseResult FPhraseInputNode< InputType >::ParsePhrase (
              TArray< FString > & InPhraseWordArray,
              FParseRecord & InParseRecord )  [override], [virtual]
```

Parses The Phrase Down This Node, Propagating Down Any Child Nodes If Required.

Parameters

InPhraseWordArray	- The Current Array of Transcription Phrases.
InParseRecord	- The Parse Record of the Current Propagation.

Returns

The Result of the Parsing of the Phrase, and any Propagation.

Reimplemented from FPhraseNode.

Definition at line 66 of file PhraseInputNode.cpp.

```
{
    if (InPhraseArray.Num() == 0)
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Emptied Phrase Array ||"))

        return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
    }

    if (MeetsInputRequirements(InPhraseArray.Last()))
    {
        // Get the Input String.
        FString InputToRecord = InPhraseArray.Pop();

        // Append the Input String to the Record.
        InParseRecord.AddPhraseString(InputToRecord);

        if (!InputToRecord.IsNumeric() && NumericParser::IsValidNumeric(InputToRecord, false))
        {
            NumericParser::StringToNumeric(InputToRecord, false);
        }

        if (!RecordInput(InputToRecord, InParseRecord))
        {
            UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Unable to Record Input ||"))

            return FParseResult(PHRASE_UNABLE_TO_PARSE, AsShared());
        }

        OnPhraseParsed.ExecuteIfBound(InParseRecord);

        return ParseChildren(InPhraseArray, InParseRecord);
    }

    return FParseResult(PHRASE_UNABLE_TO_PARSE, AsShared());
}
```

4.21.3.3 RecordInput()

```
template<typename InputType >
bool FPhraseInputNode< InputType >::RecordInput (
              const FString & InInput,
              FParseRecord & OutParseRecord )  [protected], [virtual]
```

Records the Input onto the Parse Record.

Parameters

InInput	- The Phrase To Record onto the Parse Record.
OutParseRecord	- Returns the Updated ParseRecord.

Returns

True, if the Input Was Successful in Recording. Otherwise False.

Reimplemented in FPhraseEnumInputNode< TEnum >, FPhraseEnumInputNode< EPhrase2DDirectionalInput >, FPhraseEnumInputNode< EPhraseDirectionalInput >, FPhraseEnumInputNode< EPhrasePositionalInput >, FPhraseEnumInputNode< EPhraseScrollInput >, and FPhraseStringInputNode.

Definition at line 110 of file PhraseInputNode.cpp.

```
{
    return false;
}
```

4.21.3.4 RequiresPhrase() [1/2]

```
template<typename InputType >
bool FPhraseInputNode< InputType >::RequiresPhrase (
              const FString InPhrase )  [override], [virtual]
```

Checks if the Node Requires the Given Phrase.

Parameters

InPhrase	- The Phrase To Check if Required By The Node.

Returns

True, if the Phrase is Required. Otherwise False.

Reimplemented from FPhraseNode.

Definition at line 51 of file PhraseInputNode.cpp.

```
{
    return MeetsInputRequirements(InPhrase);
}
```

4.21.3.5 RequiresPhrase() [2/2]

```
template<typename InputType >
bool FPhraseInputNode< InputType >::RequiresPhrase (
             const FString InPhrase,
             int32 & OutDistance )  [override], [virtual]
```

Checks if the Node Requires the Given Phrase, and Returns the Distance of the Phrase.

Parameters

InPhrase	- The Phrase To Check if Required By The Node.
OutDistance	- The Returned Distancing from the Target Phrase To The BoundPhrase.

Returns

True, if the Phrase is Required. Otherwise False.

Reimplemented from FPhraseNode.

Definition at line 57 of file PhraseInputNode.cpp.

```
{
    bool bMeetsRequirements = MeetsInputRequirements(InPhrase);
    OutDistance = bMeetsRequirements ? 0 : INT32_MAX;

    return bMeetsRequirements;
}
```

4.21.4 Member Data Documentation

4.21.4.1 OnInputReceived

```
template<typename InputType = int32>
TDelegate<void(InputType ReceivedInput)> FPhraseInputNode< InputType >::OnInputReceived
```

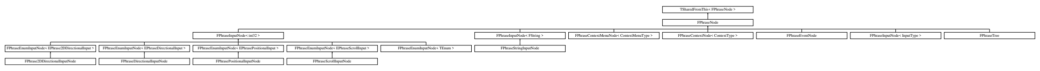
Definition at line 33 of file PhraseInputNode.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseInputNode.h
- Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseInputNode.cpp

4.22 FPhraseNode Class Reference

Inheritance diagram for FPhraseNode:


TSharedFromThis< FPhraseNode >
FPhraseNode
FPhraseInputNode< int32 >
FPhraseInputNode< FString >
FPhraseContextMenuNode< ContextMenuType >
FPhraseContextNode< ContextType >
FPhraseEventNode
FPhraseInputNode< InputType >
FPhraseTree
FPhraseEnumInputNode< EPhrase2DDirectionalInput >
FPhraseEnumInputNode< EPhraseDirectionalInput >
FPhraseEnumInputNode< EPhrasePositionalInput >
FPhraseEnumInputNode< EPhraseScrollInput >
FPhraseEnumInputNode< TEnum >
FPhraseStringInputNode
FPhrase2DDirectionalInputNode
FPhraseDirectionalInputNode
FPhrasePositionalInputNode
FPhraseScrollInputNode

Public Member Functions

- FPhraseNode (const TCHAR ∗InBoundPhrase)
- FPhraseNode (const TCHAR ∗InBoundPhrase, TDelegate< void(FParseRecord &Record)> InOnPhrase↩ Parsed)
- FPhraseNode (const TCHAR ∗InBoundPhrase, TPhraseNodeArray InChildNodes)
- FPhraseNode (const TCHAR ∗InBoundPhrase, TDelegate< void(FParseRecord &Record)> InOnPhrase↩ Parsed, TPhraseNodeArray InChildNodes)
- virtual bool IsLeafNode () const

  Checks if the Node is a Leaf Node.
- virtual bool HasLeafChild () const
- virtual bool RequiresPhrase (const FString InPhrase)

  Checks if the Node Requires the Given Phrase.
- virtual bool RequiresPhrase (const FString InPhrase, int32 &OutDistance)

  Checks if the Node Requires the Given Phrase, and Returns the Distance of the Phrase.
- virtual FParseResult ParsePhrase (TArray< FString > &InPhraseWordArray, FParseRecord &InParse↩ Record)

  Parses The Phrase Down This Node, Propagating Down Any Child Nodes If Required.
- virtual FParseResult ParsePhraseAsContext (TArray< FString > &InPhraseWordArray, FParseRecord &In↩ ParseRecord)

  Parses the Phrase Down This Node, Propagating Down Any Child Nodes If Required. Does not Pop the Phrase Array.
- virtual FParseResult ParsePhraseIfRequired (TArray< FString > &InPhraseWordArray, FParseRecord &In↩ ParseRecord)

  If the Phrase If Required, Parses the Phrase Down This Node, Propagating Down Any Child Nodes If Required.
- virtual FParseResult ParseChildren (TArray< FString > &InPhraseArray, FParseRecord &InParseRecord)

  Parses The Children Node of this Node.
- bool CanBindChild (TPhraseNode &InNode)

  Checks if the Given Node Can Be Bound as a Child Node.
- bool BindChildNode (TPhraseNode InNode)

  Binds the Given Node as a Child Node.
- bool BindChildNodeForce (TPhraseNode InNode)

  Forcefully Binds the Given Node as a Child, performing no checks.
- bool BindChildrenNodes (TPhraseNodeArray InNodes)

  Binds an Array of Nodes as Children of this Node.
- bool BindChildrenNodesForce (TPhraseNodeArray InNodes)

  Forcefully Binds an Array of Nodes as Children of this Node, performing no checks.

Public Attributes

- TWeakPtr< FPhraseNode > ParentNode

  This Nodes Parent Node.
- TPhraseNodeArray ChildNodes

  The Child Nodes of the Node.
- FString BoundPhrase

  The Phrase Bound to this
- TDelegate< void(FParseRecord &Record)> OnPhraseParsed

Protected Member Functions

- bool HasLeafChild ()

  Filters through the children, to check if it contains a Leaf Child.

Protected Attributes

- bool bHasLeafChild

  Records if the Node has a Leaf Child.

4.22.1 Detailed Description

Definition at line 54 of file PhraseNode.h.

4.22.2 Constructor & Destructor Documentation

4.22.2.1 FPhraseNode() [1/4]

```
FPhraseNode::FPhraseNode (
              const TCHAR * InBoundPhrase )
```

Definition at line 9 of file PhraseNode.cpp.

```
00010 {
00011     BoundPhrase = InBoundPhrase;
00012     BoundPhrase.ToUpperInline();
00013
00014     ChildNodes = TArray<TSharedPtr<FPhraseNode»();
00015 }
```

4.22.2.2 FPhraseNode() [2/4]

```
FPhraseNode::FPhraseNode (
            const TCHAR * InBoundPhrase,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed )
```

Definition at line 17 of file PhraseNode.cpp.

```
{
    BoundPhrase = InBoundPhrase;
    BoundPhrase.ToUpperInline();

    OnPhraseParsed = InOnPhraseParsed;
    ChildNodes = TArray<TSharedPtr<FPhraseNode»();
}
```

4.22.2.3 FPhraseNode() [3/4]

```
FPhraseNode::FPhraseNode (
            const TCHAR * InBoundPhrase,
            TPhraseNodeArray InChildNodes )
```

Definition at line 26 of file PhraseNode.cpp.

```
{
    BoundPhrase = InBoundPhrase;
    BoundPhrase.ToUpperInline();

    ChildNodes = InChildNodes;
}
```

4.22.2.4 FPhraseNode() [4/4]

```
FPhraseNode::FPhraseNode (
            const TCHAR * InBoundPhrase,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes )
```

Definition at line 34 of file PhraseNode.cpp.

```
{
    BoundPhrase = InBoundPhrase;
    BoundPhrase.ToUpperInline();

    OnPhraseParsed = InOnPhraseParsed;
    ChildNodes = InChildNodes;
}
```

4.22.2.5 ∼FPhraseNode()

```
FPhraseNode::∼FPhraseNode ( )  [virtual]
```

Definition at line 43 of file PhraseNode.cpp.

```
{

}
```

4.22.3 Member Function Documentation

4.22.3.1 BindChildNode()

```
bool FPhraseNode::BindChildNode (
            TPhraseNode InNode )
```

Binds the Given Node as a Child Node.

Parameters

InNode	- The Node To Bind as a Child of This Node.

Returns

True, if the Node was Successfully Bound. Otherwise False.

Definition at line 124 of file PhraseNode.cpp.

```
{
    if (!InNode.IsValid())
        return false;

    for (auto& ChildNode : ChildNodes)
    {
        if (ChildNode->RequiresPhrase(InNode->BoundPhrase))
        {
            return ChildNode->BindChildrenNodes(InNode->ChildNodes);
        }
        else
        {
            ChildNodes.AddUnique(ChildNode);
            return true;
        }
    }

    return false;
}
```

4.22.3.2 BindChildNodeForce()

```
bool FPhraseNode::BindChildNodeForce (
            TPhraseNode InNode )
```

Forcefully Binds the Given Node as a Child, performing no checks.

Parameters

InNode	- The Node To Foce Bind as a Child.

Returns

True, if the Node was Successfully Bound. Otherwise False.

Definition at line 145 of file PhraseNode.cpp.
```
{
    ChildNodes.AddUnique(InNode);

    return true;
}
```

4.22.3.3 BindChildrenNodes()

```
bool FPhraseNode::BindChildrenNodes (
              TPhraseNodeArray InNodes )
```

Binds an Array of Nodes as Children of this Node.

Parameters

InNodes	- The Array of Nodes To Bind as Children.

Returns

True, if the Nodes were Successfully Bound. Otherwise False.

Definition at line 152 of file PhraseNode.cpp.
```
{
    for (auto& InNode : InNodes)
    {
        for (auto& ChildNode : ChildNodes)
        {
            if (ChildNode->RequiresPhrase(InNode->BoundPhrase))
            {
                return ChildNode->BindChildrenNodes(InNode->ChildNodes);
            }
            else
            {
                ChildNodes.AddUnique(ChildNode);
                return true;
            }
        }
    }

    return false;
}
```

4.22.3.4 BindChildrenNodesForce()

```
bool FPhraseNode::BindChildrenNodesForce (
              TPhraseNodeArray InNodes )
```

Forcefully Binds an Array of Nodes as Children of this Node, performing no checks.

Parameters

InNodes	- The Array of Nodes To Bind sa Children.

Returns

True, if the Nodes were successfully bound. Otherwise False.

Definition at line 173 of file PhraseNode.cpp.

```
{
    for (auto& InNode : InNodes)
    {
        ChildNodes.AddUnique(InNode);
    }

    return true;
}
```

4.22.3.5 CanBindChild()

```
bool FPhraseNode::CanBindChild (
            TPhraseNode & InNode )
```

Checks if the Given Node Can Be Bound as a Child Node.

Parameters

InNode	- The Node To Check If It Can Be Bound.

Returns

True, if the Node Can Be Bound as a Child. Otherwise False.

Definition at line 111 of file PhraseNode.cpp.

```
{
    for (auto& ChildNode : ChildNodes)
    {
        if (ChildNode->RequiresPhrase(InNode->BoundPhrase) || ChildNode->IsLeafNode())
        {
            return false;
        }
    }

    return true;
}
```

4.22.3.6 HasLeafChild() [1/2]

```
bool FPhraseNode::HasLeafChild ( )  [protected]
```

Filters through the children, to check if it contains a Leaf Child.

Definition at line 183 of file PhraseNode.cpp.

```
{
    return ChildNodes.Num() == 1 && ChildNodes[0]->IsLeafNode();
}
```

4.22.3.7 HasLeafChild() [2/2]

```
bool FPhraseNode::HasLeafChild ( ) const  [virtual]
```

Definition at line 48 of file PhraseNode.cpp.

```
00049 {
00050     return bHasLeafChild;
00051 }
```

4.22.3.8 IsLeafNode()

```
virtual bool FPhraseNode::IsLeafNode ( ) const  [inline], [virtual]
```

Checks if the Node is a Leaf Node.

Returns

True, if the Node is a Leaf Node. Otherwise False.

Reimplemented in FPhraseEventNode.

Definition at line 69 of file PhraseNode.h.

```
00069 { return false; }
```

4.22.3.9 ParseChildren()

```
FParseResult FPhraseNode::ParseChildren (
             TArray< FString > & InPhraseArray,
             FParseRecord & InParseRecord )  [virtual]
```

Parses The Children Node of this Node.

Parameters

InPhraseArray	- The Current Array of Transcription Phrases.
InParseRecord	- The Parse Record of the Current Propagation.

Returns

The Result of the Parsing of the Phrase, and any Propagation.

Definition at line 188 of file PhraseNode.cpp.

```
00189 {
00190     if (HasLeafChild())
00191         return ChildNodes[0]->ParsePhrase(InPhraseArray, InParseRecord);
00192     if (InPhraseArray.IsEmpty())
00193         return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
00194
00195     // Below Can Be Optimized.
00196     // Maybe bypass the loop if Distance == 0 and Sort ChildNodes with Derrived PhraseNodes Last?
00197
```

```
    int FoundChildIndex = -1;
    {
        int32 FoundChildDistance = INT32_MAX, CurrentDistance = INT32_MAX;

        for (int i = 0; i < ChildNodes.Num(); i++)
        {
            // Child Nodes Require Unique Phrases to Siblings.
            if (ChildNodes[i]->RequiresPhrase(InPhraseArray.Last(), CurrentDistance))
            {
                if (FoundChildDistance > CurrentDistance)
                {
                    FoundChildIndex = i;
                    FoundChildDistance = CurrentDistance;
                }
            }
        }
    }

    if (FoundChildIndex != -1)
    {
        return ChildNodes[FoundChildIndex]->ParsePhrase(InPhraseArray, InParseRecord);
    }

    /*else if (!InPhraseArray.IsEmpty())
    {
        return FParseResult(PHRASE_REQUIRES_MORE_CORRECT_PHRASES, AsShared());
    }*/

    return FParseResult(PHRASE_UNABLE_TO_PARSE, AsShared());
}
```

4.22.3.10 ParsePhrase()

```
FParseResult FPhraseNode::ParsePhrase (
            TArray< FString > & InPhraseWordArray,
            FParseRecord & InParseRecord )  [virtual]
```

Parses The Phrase Down This Node, Propagating Down Any Child Nodes If Required.

Parameters

InPhraseWordArray	- The Current Array of Transcription Phrases.
InParseRecord	- The Parse Record of the Current Propagation.

Returns

The Result of the Parsing of the Phrase, and any Propagation.

Reimplemented in FPhraseEventNode, FPhraseInputNode< InputType >, FPhraseInputNode< int32 >, FPhraseInputNode< FString >, FPhraseTree, FPhraseContextMenuNode< ContextMenuType >, and FPhraseContextNode< Conte

Definition at line 65 of file PhraseNode.cpp.

```
                                                            {
    if (InPhraseArray.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Emptied Phrase Array ||"))

        return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
    }

    // Pop the Phrase Linked to this Node.
    // Apply to the Record.
    FString LinkedPhrase = InPhraseArray.Pop();

    // Append Removed Phrase To Record.
    InParseRecord.AddPhraseString(LinkedPhrase);

    OnPhraseParsed.ExecuteIfBound(InParseRecord);

    // Pass
    return ParseChildren(InPhraseArray, InParseRecord);
}
```

4.22.3.11 ParsePhraseAsContext()

```
FParseResult FPhraseNode::ParsePhraseAsContext (
            TArray< FString > & InPhraseWordArray,
            FParseRecord & InParseRecord )  [virtual]
```

Parses the Phrase Down This Node, Propagating Down Any Child Nodes If Required. Does not Pop the Phrase Array.

Parameters

InPhraseWordArray	- The Current Array of Transcription Phrases.
InParseRecord	- The Parse Record of the Current Propagation.

Returns

The Result of the Parsing of the Phrase, and any Propagation.

Reimplemented in FPhraseContextMenuNode< ContextMenuType >, and FPhraseContextNode< ContextType >.

Definition at line 87 of file PhraseNode.cpp.

```
{
    if (InPhraseWordArray.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Emptied Phrase Array ||"))

            return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
    }

    OnPhraseParsed.ExecuteIfBound(InParseRecord);

    return ParseChildren(InPhraseWordArray, InParseRecord);
}
```

4.22.3.12 ParsePhraseIfRequired()

```
FParseResult FPhraseNode::ParsePhraseIfRequired (
            TArray< FString > & InPhraseWordArray,
            FParseRecord & InParseRecord )  [virtual]
```

If the Phrase If Required, Parses the Phrase Down This Node, Propagating Down Any Child Nodes If Required.

Definition at line 101 of file PhraseNode.cpp.

```
{
    if (RequiresPhrase(InPhraseWordArray.Last()))
    {
        return ParsePhrase(InPhraseWordArray, InParseRecord);
    }

    return FParseResult(PHRASE_UNABLE_TO_PARSE);
}
```

4.22.3.13 RequiresPhrase() [1/2]

```
bool FPhraseNode::RequiresPhrase (
             const FString InPhrase )  [virtual]
```

Checks if the Node Requires the Given Phrase.

Parameters

InPhrase	- The Phrase To Check if Required By The Node.

Returns

True, if the Phrase is Required. Otherwise False.

Reimplemented in FPhraseEventNode, FPhraseInputNode< InputType >, FPhraseInputNode< int32 >, and FPhraseInputNode< FString >.

Definition at line 53 of file PhraseNode.cpp.

```
{
    return InPhrase.Equals(BoundPhrase, ESearchCase::IgnoreCase) ||
      Algo::LevenshteinDistance(BoundPhrase, InPhrase) < 3;
}
```

4.22.3.14 RequiresPhrase() [2/2]

```
bool FPhraseNode::RequiresPhrase (
             const FString InPhrase,
             int32 & OutDistance )  [virtual]
```

Checks if the Node Requires the Given Phrase, and Returns the Distance of the Phrase.

Parameters

InPhrase	- The Phrase To Check if Required By The Node.
OutDistance	- The Returned Distancing from the Target Phrase To The BoundPhrase.

Returns

True, if the Phrase is Required. Otherwise False.

Reimplemented in FPhraseEventNode, FPhraseInputNode< InputType >, FPhraseInputNode< int32 >, and FPhraseInputNode< FString >.

Definition at line 58 of file PhraseNode.cpp.

```
{
    OutDistance = Algo::LevenshteinDistance(BoundPhrase, InPhrase);

    return InPhrase.Equals(BoundPhrase, ESearchCase::IgnoreCase) || OutDistance < 3;
}
```

4.22.4 Member Data Documentation

4.22.4.1 bHasLeafChild

```
bool FPhraseNode::bHasLeafChild  [protected]
```

Records if the Node has a Leaf Child.

Definition at line 185 of file PhraseNode.h.

4.22.4.2 BoundPhrase

```
FString FPhraseNode::BoundPhrase
```

The Phrase Bound to this

Definition at line 175 of file PhraseNode.h.

4.22.4.3 ChildNodes

```
TPhraseNodeArray FPhraseNode::ChildNodes
```

The Child Nodes of the Node.

Definition at line 170 of file PhraseNode.h.

4.22.4.4 OnPhraseParsed

```
TDelegate<void (FParseRecord& Record)> FPhraseNode::OnPhraseParsed
```

Definition at line 178 of file PhraseNode.h.

4.22.4.5 ParentNode

```
TWeakPtr<FPhraseNode> FPhraseNode::ParentNode
```

This Nodes Parent Node.

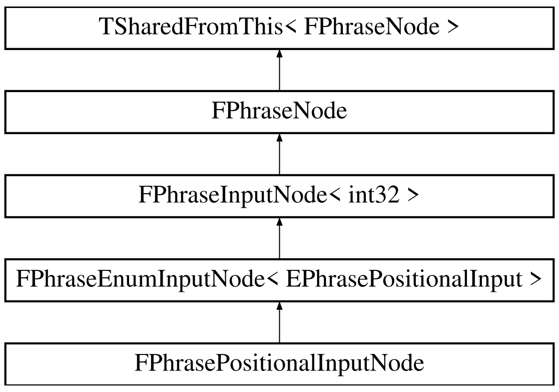
Definition at line 165 of file PhraseNode.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseNode.h
- Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseNode.cpp

4.23 FPhrasePositionalInputNode Class Reference

Inheritance diagram for FPhrasePositionalInputNode:


TSharedFromThis< FPhraseNode >
FPhraseNode
FPhraseInputNode< int32 >
FPhraseEnumInputNode< EPhrasePositionalInput >
FPhrasePositionalInputNode

Public Member Functions

- FPhrasePositionalInputNode (const TCHAR ∗NodeName)
- FPhrasePositionalInputNode (const TCHAR ∗NodeName, TPhraseNodeArray InChildNodes)
- FPhrasePositionalInputNode (const TCHAR ∗NodeName, TDelegate< void(FParseRecord &Record)> In↩ OnPhraseParsed, TPhraseNodeArray InChildNodes)
- FPhrasePositionalInputNode (const TCHAR ∗NodeName, TPhraseNodeArray InChildNodes, TDelegate< void(int32 Input)> InOnInputRecieved)
- FPhrasePositionalInputNode (const TCHAR ∗NodeName, TDelegate< void(FParseRecord &Record)> In↩ OnPhraseParsed, TPhraseNodeArray InChildNodes, TDelegate< void(int32 Input)> InOnInputRecieved)

Additional Inherited Members

4.23.1 Detailed Description

Definition at line 80 of file PhraseDirectionalInputNode.h.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 FPhrasePositionalInputNode() [1/5]

```
FPhrasePositionalInputNode::FPhrasePositionalInputNode (
            const TCHAR ∗ NodeName )  [inline]
```

Definition at line 83 of file PhraseDirectionalInputNode.h.

```
00084        : FPhraseEnumInputNode<EPhrasePositionalInput>(NodeName)
00085    {}
```

4.23.2.2 FPhrasePositionalInputNode() [2/5]

```
FPhrasePositionalInputNode::FPhrasePositionalInputNode (
            const TCHAR * NodeName,
            TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 87 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhrasePositionalInput>(NodeName, InChildNodes)
    {}
```

4.23.2.3 FPhrasePositionalInputNode() [3/5]

```
FPhrasePositionalInputNode::FPhrasePositionalInputNode (
            const TCHAR * NodeName,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 91 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhrasePositionalInput>(NodeName, InOnPhraseParsed, InChildNodes)
    {}
```

4.23.2.4 FPhrasePositionalInputNode() [4/5]

```
FPhrasePositionalInputNode::FPhrasePositionalInputNode (
            const TCHAR * NodeName,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(int32 Input)> InOnInputRecieved )  [inline]
```

Definition at line 95 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhrasePositionalInput>(NodeName, InChildNodes, InOnInputRecieved)
    {}
```

4.23.2.5 FPhrasePositionalInputNode() [5/5]

```
FPhrasePositionalInputNode::FPhrasePositionalInputNode (
            const TCHAR * NodeName,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(int32 Input)> InOnInputRecieved )  [inline]
```

Definition at line 99 of file PhraseDirectionalInputNode.h.

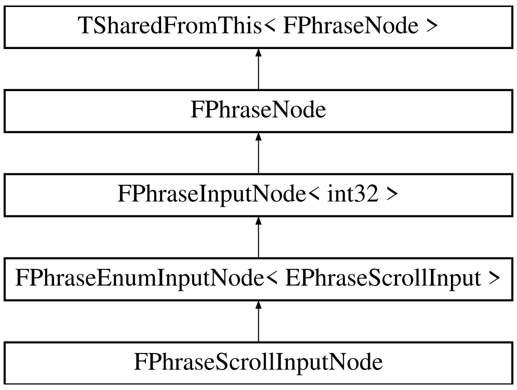
```
        : FPhraseEnumInputNode<EPhrasePositionalInput>(NodeName, InOnPhraseParsed, InChildNodes,
    InOnInputRecieved)
    {}
```

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseDirectionalInputNode.h

4.24 FPhraseScrollInputNode Class Reference

Inheritance diagram for FPhraseScrollInputNode:


TSharedFromThis< FPhraseNode >
FPhraseNode
FPhraseInputNode< int32 >
FPhraseEnumInputNode< EPhraseScrollInput >
FPhraseScrollInputNode

Public Member Functions

- FPhraseScrollInputNode (const TCHAR ∗NodeName)
- FPhraseScrollInputNode (const TCHAR ∗NodeName, TPhraseNodeArray InChildNodes)
- FPhraseScrollInputNode (const TCHAR ∗NodeName, TDelegate< void(FParseRecord &Record)> InOn↩ PhraseParsed, TPhraseNodeArray InChildNodes)
- FPhraseScrollInputNode (const TCHAR ∗NodeName, TPhraseNodeArray InChildNodes, TDelegate< void(int32 Input)> InOnInputRecieved)
- FPhraseScrollInputNode (const TCHAR ∗NodeName, TDelegate< void(FParseRecord &Record)> InOn↩ PhraseParsed, TPhraseNodeArray InChildNodes, TDelegate< void(int32 Input)> InOnInputRecieved)

Additional Inherited Members

4.24.1 Detailed Description

Definition at line 56 of file PhraseDirectionalInputNode.h.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 FPhraseScrollInputNode() [1/5]

```
FPhraseScrollInputNode::FPhraseScrollInputNode (
              const TCHAR * NodeName )  [inline]
```

Definition at line 59 of file PhraseDirectionalInputNode.h.

```
00060         : FPhraseEnumInputNode<EPhraseScrollInput>(NodeName)
00061     {}
```

4.24.2.2 FPhraseScrollInputNode() [2/5]

```
FPhraseScrollInputNode::FPhraseScrollInputNode (
            const TCHAR * NodeName,
            TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 63 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhraseScrollInput>(NodeName, InChildNodes)
    {}
```

4.24.2.3 FPhraseScrollInputNode() [3/5]

```
FPhraseScrollInputNode::FPhraseScrollInputNode (
            const TCHAR * NodeName,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes )  [inline]
```

Definition at line 67 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhraseScrollInput>(NodeName, InOnPhraseParsed, InChildNodes)
    {}
```

4.24.2.4 FPhraseScrollInputNode() [4/5]

```
FPhraseScrollInputNode::FPhraseScrollInputNode (
            const TCHAR * NodeName,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(int32 Input)> InOnInputRecieved )  [inline]
```

Definition at line 71 of file PhraseDirectionalInputNode.h.

```
        : FPhraseEnumInputNode<EPhraseScrollInput>(NodeName, InChildNodes, InOnInputRecieved)
    {}
```

4.24.2.5 FPhraseScrollInputNode() [5/5]

```
FPhraseScrollInputNode::FPhraseScrollInputNode (
            const TCHAR * NodeName,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(int32 Input)> InOnInputRecieved )  [inline]
```

Definition at line 75 of file PhraseDirectionalInputNode.h.

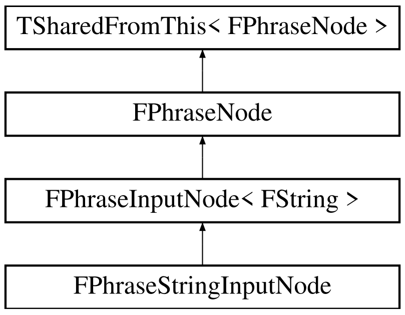
```
        : FPhraseEnumInputNode<EPhraseScrollInput>(NodeName, InOnPhraseParsed, InChildNodes,
      InOnInputRecieved)
    {}
```

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseDirectionalInputNode.h

4.25 FPhraseStringInputNode Class Reference

Inheritance diagram for FPhraseStringInputNode:



Public Member Functions

- FPhraseStringInputNode (const TCHAR ∗InInputString)
- FPhraseStringInputNode (const TCHAR ∗InInputString, TPhraseNodeArray InChildNodes)
- FPhraseStringInputNode (const TCHAR ∗InInputString, TDelegate< void(FParseRecord &Record)> InOn↩ PhraseParsed, TPhraseNodeArray InChildNodes)
- FPhraseStringInputNode (const TCHAR ∗InInputString, TPhraseNodeArray InChildNodes, TDelegate< void(FString Input)> InOnInputRecieved)
- **FPhraseStringInputNode** (const TCHAR ∗InInputString, TDelegate< void(FParseRecord &Record)> In↩ OnPhraseParsed, TPhraseNodeArray InChildNodes, TDelegate< void(FString Input)> InOnInputRecieved)

Protected Member Functions

- virtual bool MeetsInputRequirements (const FString &InPhrase) override

  Checks if the Given Phrase Meets Requirements for usage as Input. In Correlation to this Nodes Input Specifications.
- virtual bool RecordInput (const FString &InInput, FParseRecord &OutParseRecord) override

  Records the Input onto the Parse Record.

Additional Inherited Members

4.25.1 Detailed Description

Definition at line 11 of file PhraseStringInputNode.h.

4.25.2 Constructor & Destructor Documentation

4.25.2.1 FPhraseStringInputNode() [1/4]

```
FPhraseStringInputNode::FPhraseStringInputNode (
            const TCHAR * InInputString )
```

Definition at line 7 of file PhraseStringInputNode.cpp.

```
    : FPhraseInputNode(InInputString)
{

};
```

4.25.2.2 FPhraseStringInputNode() [2/4]

```
FPhraseStringInputNode::FPhraseStringInputNode (
            const TCHAR * InInputString,
            TPhraseNodeArray InChildNodes )
```

Definition at line 13 of file PhraseStringInputNode.cpp.

```
    : FPhraseInputNode(InInputString, InChildNodes)
{

}
```

4.25.2.3 FPhraseStringInputNode() [3/4]

```
FPhraseStringInputNode::FPhraseStringInputNode (
            const TCHAR * InInputString,
            TDelegate< void(FParseRecord &Record)> InOnPhraseParsed,
            TPhraseNodeArray InChildNodes )
```

Definition at line 19 of file PhraseStringInputNode.cpp.

```
    : FPhraseInputNode(InInputString, InOnPhraseParse, InChildNodes)
{

}
```

4.25.2.4 FPhraseStringInputNode() [4/4]

```
FPhraseStringInputNode::FPhraseStringInputNode (
            const TCHAR * InInputString,
            TPhraseNodeArray InChildNodes,
            TDelegate< void(FString Input)> InOnInputRecieved )
```

Definition at line 25 of file PhraseStringInputNode.cpp.

```
    : FPhraseInputNode(InInputString, InChildNodes, InOnInputRecieved)
{

}
```

4.25.2.5 ∼FPhraseStringInputNode()

```
FPhraseStringInputNode::∼FPhraseStringInputNode ( )
```

Definition at line 31 of file PhraseStringInputNode.cpp.

```
{
}
```

4.25.3 Member Function Documentation

4.25.3.1 MeetsInputRequirements()

```
bool FPhraseStringInputNode::MeetsInputRequirements (
            const FString & InPhrase )  [override], [protected], [virtual]
```

Checks if the Given Phrase Meets Requirements for usage as Input. In Correlation to this Nodes Input Specifications.

Parameters

InPhrase	- The Phrase To Check If It Meets Requirements.

Returns

True, if the Phrase Meets Requirements. Otherwise False.

Reimplemented from FPhraseInputNode< FString >.

Definition at line 36 of file PhraseStringInputNode.cpp.

```
{
    if (InPhrase.IsEmpty())
        return false;
    else return true;
}
```

4.25.3.2 RecordInput()

```
bool FPhraseStringInputNode::RecordInput (
            const FString & InInput,
            FParseRecord & OutParseRecord )  [override], [protected], [virtual]
```

Records the Input onto the Parse Record.

Parameters

InInput	- The Phrase To Record onto the Parse Record.
OutParseRecord	- Returns the Updated ParseRecord.

Returns

True, if the Input Was Successful in Recording. Otherwise False.

Reimplemented from FPhraseInputNode< FString >.

Definition at line 43 of file PhraseStringInputNode.cpp.

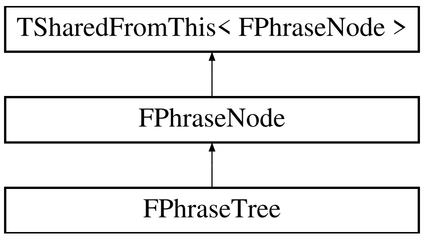
```
{
    if (InInput.IsEmpty())
        return false;

    UParseStringInput* ParseInput = MakeParseInput<UParseStringInput>();
    ParseInput->SetValue(InInput);

    OutParseRecord.AddPhraseInput(BoundPhrase, ParseInput);

    OnInputReceived.ExecuteIfBound(InInput);

    return true;
}
```

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseStringInputNode.h
- Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseStringInputNode.cpp

4.26 FPhraseTree Class Reference

Inheritance diagram for FPhraseTree:



Public Member Functions

- FPhraseTreeContextManager & GetContextManager ()
- bool Tick (float DeltaTime)
- virtual FParseResult ParsePhrase (TArray< FString > &InPhraseWordArray, FParseRecord &InParse↩ Record) override

  Parses The Phrase Down This Node, Propagating Down Any Child Nodes If Required.
- void BindBranch (const TPhraseNode &InNode)

  Bind a branch to the tree. Attaching to any overlapping nodes.
- void BindBranches (const TPhraseNodeArray &InNodes)

  Bind Multiple Branches to the Tree, that are not connected.
- void ParseTranscription (TArray< FString > InTranscriptionSegments)

  Parses and Propogates the given Transcription Segments down the tree.

Additional Inherited Members

4.26.1 Detailed Description

Definition at line 227 of file PhraseTree.h.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 FPhraseTree()

```
FPhraseTree::FPhraseTree ( )
```

Definition at line 12 of file PhraseTree.cpp.

```
00012                           : FPhraseNode(TEXT("ROOT_NODE"))
00013 {
00014     ContextManager = FPhraseTreeContextManager();
00015
00016     FTickerDelegate TickDelegate = FTickerDelegate::CreateRaw(this, &FPhraseTree::Tick);
00017     TickDelegateHandle = FTSTicker::GetCoreTicker().AddTicker(TickDelegate);
00018 }
```

4.26.2.2 ∼FPhraseTree()

```
FPhraseTree::∼FPhraseTree ( )
```

Definition at line 20 of file PhraseTree.cpp.

```
00021 {
00022     FTSTicker::GetCoreTicker().RemoveTicker(TickDelegateHandle);
00023 }
```

4.26.3 Member Function Documentation

4.26.3.1 BindBranch()

```
void FPhraseTree::BindBranch (
              const TPhraseNode & InNode )
```

Bind a branch to the tree. Attaching to any overlapping nodes.

Parameters

InNode	The constructed branch to attach to the tree.

Definition at line 182 of file PhraseTree.cpp.

```
{
    TArray<FPhraseTreeBranchBind> ToBindArray = TArray<FPhraseTreeBranchBind>();

    ToBindArray.Add(FPhraseTreeBranchBind(AsShared(), InNode));

    while (!ToBindArray.IsEmpty())
    {
        FPhraseTreeBranchBind BranchToBind = ToBindArray.Pop();

        // Check all ChildNodes to see if they are similar in purpose.
        for (auto& ChildNode : BranchToBind.StartNode->ChildNodes)
        {
            // If a ChildNode meets the same requirements as the BranchRoot,
            // then Split Bind Process to the ChildNodes.
            if (ChildNode->RequiresPhrase(BranchToBind.BranchRoot->BoundPhrase))
            {
                for (auto& BranchChildNode : BranchToBind.BranchRoot->ChildNodes)
                {
                    ToBindArray.Add(FPhraseTreeBranchBind(ChildNode, BranchChildNode));
                }

                continue;
            }
        }

        // If the Start Node has no similar children, then bind the branch to the start node.
        // Can force bind, as previous checks show no child is similar.
        BranchToBind.StartNode->BindChildNodeForce(BranchToBind.BranchRoot);
    }
}
```

4.26.3.2 BindBranches()

```
void FPhraseTree::BindBranches (
                const TPhraseNodeArray & InNodes )
```

Bind Multiple Branches to the Tree, that are not connected.

Definition at line 214 of file PhraseTree.cpp.

```
{
    for (const TSharedPtr<FPhraseNode>& Node : InNodes)
    {
        BindBranch(Node);
    }
}
```

4.26.3.3 GetContextManager()

```
FPhraseTreeContextManager & FPhraseTree::GetContextManager ( )  [inline]
```

Definition at line 233 of file PhraseTree.h.

```
                                            {
        return ContextManager;
    }
```

4.26.3.4 ParsePhrase()

```
FParseResult FPhraseTree::ParsePhrase (
                TArray< FString > & InPhraseWordArray,
                FParseRecord & InParseRecord )  [override], [virtual]
```

Parses The Phrase Down This Node, Propagating Down Any Child Nodes If Required.

Parameters

InPhraseWordArray	- The Current Array of Transcription Phrases.
InParseRecord	- The Parse Record of the Current Propagation.

Returns

The Result of the Parsing of the Phrase, and any Propagation.

Reimplemented from FPhraseNode.

Definition at line 141 of file PhraseTree.cpp.

```
{
    if (InPhraseWordArray.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Phrase Tree || Provided Transcription
        Segment is Empty ||"));

        return FParseResult(PHRASE_NOT_PARSED);
    }

    // First give the last visited node a chance to parse the phrase.
    // due to the possibility of connecting phrases over different transcription segments.
    if (LastVistedNode != nullptr && LastVistedNode.IsValid())
    {
        TArray<FString> PhraseWordArrayCopy = TArray(InPhraseWordArray);

        FParseResult ParseResult = LastVistedNode->ParseChildren(PhraseWordArrayCopy,
        LastVistedParseRecord);
        if (ParseResult.Result == PHRASE_PARSED)
        {
            LastVistedNode.Reset();
            InParseRecord = LastVistedParseRecord;
            LastVistedParseRecord = FParseRecord();

            return ParseResult;
        }
        else if (ParseResult.Result != PHRASE_UNABLE_TO_PARSE)
        {
            return ParseResult;
        }
    }

    // Check if the Context Stack has Objects, if so propagation from the Context Root.
    if (ContextManager.HasContextObjects())
    {
        // Propagate from the Context Root, that is the Top of the Context Stack.
        return
        ContextManager.PeekContextObject()->GetContextRoot()->ParsePhraseAsContext(InPhraseWordArray,
        InParseRecord);
    }

    // Otherwise, start a new propagation entirely from the Tree Root.
    return ParseChildren(InPhraseWordArray, InParseRecord);
}
```

4.26.3.5 ParseTranscription()

```
void FPhraseTree::ParseTranscription (
            TArray< FString > InTranscriptionSegments )
```

Parses and Propogates the given Transcription Segments down the tree.

Parameters

InTranscriptionSegments	

Definition at line 33 of file PhraseTree.cpp.

```
{
    if (InTranscriptionSegments.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Phrase Tree || Provided Transcription is Empty ||"))
        return;
    }

    TArray<FString> SegmentWordArray = TArray<FString>();
    int SegmentCount = 0;

    // Loop over any Transcription Segments.
    for (FString& TranscriptionSegment : InTranscriptionSegments)
    {
        if (TranscriptionSegment.IsEmpty())
        {
            UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Phrase Tree || Transcription Segment is Empty ||"))
            continue;
        }

        // Filter the Transcription Segment, to remove any unwanted characters.
        TranscriptionSegment.TrimStartAndEndInline();
        TranscriptionSegment.ReplaceInline(TEXT("."), TEXT(""), ESearchCase::IgnoreCase);
        TranscriptionSegment.ReplaceInline(TEXT(","), TEXT(""), ESearchCase::IgnoreCase);
        TranscriptionSegment.ToUpperInline();

        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Phrase Tree || Filtered Transcription Segment: { %s } ||"), *TranscriptionSegment)

        // Parse the Transcription Segment into an Array of Words, removing any white space.
        TranscriptionSegment.ParseIntoArrayWS(SegmentWordArray);
        if (SegmentWordArray.Num() == 0)
        {
            UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Phrase Tree || Transcription Segment has no Word Content ||"))
            continue;
        }

        Algo::Reverse(SegmentWordArray);

        // Loop until the Segment is Empty
        while (!SegmentWordArray.IsEmpty())
        {

            FParseRecord ParseRecord = FParseRecord(ContextManager.GetContextStack());
            FParseResult ParseResult = ParsePhrase(SegmentWordArray, ParseRecord);

            ContextManager.UpdateContextStack(ParseRecord.ContextObjectStack);

            UE_LOGFMT(LogOpenAccessibilityCom, Log, "|| Phrase Tree || Segment: {0} | Result: {1} ||", SegmentCount, ParseResult.Result);

            switch (ParseResult.Result)
            {
                case PHRASE_PARSED:
                case PHRASE_PARSED_AND_EXECUTED:
                {
                    OA_LOG(LogOpenAccessibilityCom, Log, TEXT("PhraseTree Propagation"), TEXT("{Success} Phrase Tree Parsed Correctly (%s)"),
                        *ParseRecord.GetPhraseString())

                    LastVistedNode.Reset();
                    LastVistedParseRecord = FParseRecord();

                    break;
                }

                case PHRASE_REQUIRES_MORE:
                {
                    OA_LOG(LogOpenAccessibilityCom, Log, TEXT("PhraseTree Propagation"), TEXT("{Failed} Phrase Tree Propagation Requires More Segments. (%s)"),
                        *ParseRecord.GetPhraseString());

                    // Store Reach Nodes, and the ParseRecord for future propagation attempts.
                    LastVistedNode = ParseResult.ReachedNode;
                    LastVistedParseRecord = ParseRecord;
                }

                case PHRASE_REQUIRES_MORE_CORRECT_PHRASES:
                {
                    OA_LOG(LogOpenAccessibilityCom, Log, TEXT("PhraseTree Propagation"), TEXT("{Failed} Phrase Tree Propagation Requires More Correct Segments. (%s)"),
                        *ParseRecord.GetPhraseString())

                    LastVistedNode = ParseResult.ReachedNode;
                    LastVistedParseRecord = ParseRecord;

                    // Dirty Way of Ensuring all Segments in Transcription are Attempted.
                    if (!SegmentWordArray.IsEmpty())
                        SegmentWordArray.Pop();

                    break;
                }

                default:
                case PHRASE_UNABLE_TO_PARSE:
                {
                    OA_LOG(LogOpenAccessibilityCom, Log, TEXT("PhraseTree Propagation"),
       TEXT("{Failed} Phrase Tree Propagation Failed. (%s)"),
                        *ParseRecord.GetPhraseString())

                    // Dirty Way of Ensuring all Segments in Transcription are Attempted.
                    if (!SegmentWordArray.IsEmpty())
                        SegmentWordArray.Pop();

                    break;
                }
            }
        }

        SegmentCount++;
        SegmentWordArray.Reset();
    }
}
```

4.26.3.6 Tick()

```
bool FPhraseTree::Tick (
            float DeltaTime )
```

Definition at line 25 of file PhraseTree.cpp.

```
{
    // Filter InActive Context Objects out of the stack.
    ContextManager.FilterContextStack();

    return true;
}
```

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree.h
- Source/OpenAccessibilityCommunication/Private/PhraseTree.cpp

4.27 FPhraseTreeBranchBind Struct Reference

Public Member Functions

- FPhraseTreeBranchBind (TPhraseNode InRootNode, TPhraseNode InBranchRoot)

Public Attributes

- TPhraseNode StartNode

  The Node to start the binding of this branch root.
- TPhraseNode BranchRoot

  The Root Node of the Branch that needs to be bound.

4.27.1 Detailed Description

Definition at line 25 of file PhraseTree.h.

4.27.2 Constructor & Destructor Documentation

4.27.2.1 FPhraseTreeBranchBind() [1/2]

```
FPhraseTreeBranchBind::FPhraseTreeBranchBind ( )  [inline]
```

Definition at line 27 of file PhraseTree.h.

```
    {
    }
```

4.27.2.2 FPhraseTreeBranchBind() [2/2]

```
FPhraseTreeBranchBind::FPhraseTreeBranchBind (
              TPhraseNode InRootNode,
              TPhraseNode InBranchRoot )  [inline]
```

Definition at line 32 of file PhraseTree.h.

```
    {
        StartNode = InRootNode;
        BranchRoot = InBranchRoot;
    }
```

4.27.2.3 ∼FPhraseTreeBranchBind()

```
FPhraseTreeBranchBind::∼FPhraseTreeBranchBind ( )  [inline]
```

Definition at line 38 of file PhraseTree.h.

```
    {
        StartNode.Reset();
        BranchRoot.Reset();
    }
```

4.27.3 Member Data Documentation

4.27.3.1 BranchRoot

```
TPhraseNode FPhraseTreeBranchBind::BranchRoot
```

The Root Node of the Branch that needs to be bound.

Definition at line 52 of file PhraseTree.h.

4.27.3.2 StartNode

```
TPhraseNode FPhraseTreeBranchBind::StartNode
```

The Node to start the binding of this branch root.

Definition at line 47 of file PhraseTree.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree.h

4.28 FPhraseTreeContextManager Struct Reference

Public Member Functions

- void IsEmpty ()

  Is the Context Stack Empty.
- bool HasContextObjects ()

  Does the Context Stack Contain Any Context Objects.
- bool HasContextObject (UPhraseTreeContextObject ∗InContextObject)

  Does the Context Stack Contain The Given Context Object.
- TArray< UPhraseTreeContextObject ∗ > GetContextStack ()

  Gets the Entire Context Stack.
- void PeekContextObject (UPhraseTreeContextObject ∗OutContextObject)

  Peeks the Top Context Object On The Stack.
- UPhraseTreeContextObject ∗ PeekContextObject ()

  Peeks the Top Context Object On The Stack.
- void PushContextObject (UPhraseTreeContextObject ∗InContextObject)

  Pushes a Context Object onto the Stack.
- void PopContextObject ()

  Pops the Top Context Object From The Stack.
- template<class CastToContextType >
  void PopContextObject (CastToContextType ∗OutContextObject)

  Pops the Top Context Object From The Stack.
- void PopContextObject (UPhraseTreeContextObject ∗OutContextObject)

  Pops the Top Context Object From The Stack.

Friends

- class FPhraseTree

4.28.1 Detailed Description

Definition at line 55 of file PhraseTree.h.

4.28.2 Constructor & Destructor Documentation

4.28.2.1 FPhraseTreeContextManager()

```
FPhraseTreeContextManager::FPhraseTreeContextManager ( )  [inline]
```

Definition at line 61 of file PhraseTree.h.

```
{

}
```

4.28.2.2 ∼FPhraseTreeContextManager()

```
FPhraseTreeContextManager::∼FPhraseTreeContextManager ( )  [inline]
```

Definition at line 66 of file PhraseTree.h.

```
{

}
```

4.28.3 Member Function Documentation

4.28.3.1 GetContextStack()

```
TArray< UPhraseTreeContextObject * > FPhraseTreeContextManager::GetContextStack ( )  [inline]
```

Gets the Entire Context Stack.

Returns

An Array Containing the Current Context Stack.

Definition at line 104 of file PhraseTree.h.

```
{
    return this->ContextObjectStack;
}
```

4.28.3.2 HasContextObject()

```
bool FPhraseTreeContextManager::HasContextObject (
            UPhraseTreeContextObject * InContextObject )  [inline]
```

Does the Context Stack Contain The Given Context Object.

Parameters

InContextObject	- The Context Object To Check if On The Stack.

Returns

True, if the Context Object is Contained on the Stack.

Definition at line 95 of file PhraseTree.h.

```
{
    return this->ContextObjectStack.Contains(InContextObject);
}
```

4.28.3.3 HasContextObjects()

```
bool FPhraseTreeContextManager::HasContextObjects ( )  [inline]
```

Does the Context Stack Contain Any Context Objects.

Returns

True, if Context Objects are on the stack. Otherwise False.

Definition at line 85 of file PhraseTree.h.

```
{
    return this->ContextObjectStack.Num() > 0;
}
```

4.28.3.4 IsEmpty()

```
void FPhraseTreeContextManager::IsEmpty ( )  [inline]
```

Is the Context Stack Empty.

Definition at line 76 of file PhraseTree.h.

```
{
    this->ContextObjectStack.IsEmpty();
}
```

4.28.3.5 PeekContextObject() [1/2]

```
UPhraseTreeContextObject * FPhraseTreeContextManager::PeekContextObject ( )  [inline]
```

Peeks the Top Context Object On The Stack.

Returns

The Top Context Object on the Stack.

Definition at line 124 of file PhraseTree.h.

```
{
    return this->ContextObjectStack.Top();
}
```

4.28.3.6 PeekContextObject() [2/2]

```
void FPhraseTreeContextManager::PeekContextObject (
            UPhraseTreeContextObject * OutContextObject )  [inline]
```

Peeks the Top Context Object On The Stack.

Parameters

OutContextObject	- Returns the Top Context Object.

Definition at line 115 of file PhraseTree.h.

```
{
    OutContextObject = this->ContextObjectStack.Top();
}
```

4.28.3.7 PopContextObject() [1/3]

```
void FPhraseTreeContextManager::PopContextObject ( )  [inline]
```

Pops the Top Context Object From The Stack.

Definition at line 141 of file PhraseTree.h.

```
{
    this->ContextObjectStack.Pop();
}
```

4.28.3.8 PopContextObject() [2/3]

```
template<class CastToContextType >
void FPhraseTreeContextManager::PopContextObject (
            CastToContextType * OutContextObject )  [inline]
```

Pops the Top Context Object From The Stack.

Template Parameters

CastToContextType	DownCast Type for the Popped Context Object. (Must be Derrived From UPhraseTreeContextObject).

Parameters

OutContextObject	- Returns the Popped Downcasted Context Object From the Stack.

Definition at line 152 of file PhraseTree.h.

```
{
    OutContextObject = Cast<CastToContextType>(this->ContextObjectStack.Pop());
}
```

4.28.3.9 PopContextObject() [3/3]

```
void FPhraseTreeContextManager::PopContextObject (
            UPhraseTreeContextObject * OutContextObject )  [inline]
```

Pops the Top Context Object From The Stack.

Parameters

OutContextObject	- Returns the Popped Context Object From the Stack.

Definition at line 161 of file PhraseTree.h.

```
{
    OutContextObject = this->ContextObjectStack.Pop();
}
```

4.28.3.10 PushContextObject()

```
void FPhraseTreeContextManager::PushContextObject (
            UPhraseTreeContextObject * InContextObject )  [inline]
```

Pushes a Context Object onto the Stack.

Parameters

InContextObject	- The Context Object To Push Onto The Stack.

Definition at line 133 of file PhraseTree.h.

```
{
    this->ContextObjectStack.Push(InContextObject);
}
```

4.28.4 Friends And Related Function Documentation

4.28.4.1 FPhraseTree

```
friend class FPhraseTree  [friend]
```

Definition at line 57 of file PhraseTree.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree.h

4.29 FSocketCommunicationServer Class Reference

Public Member Functions

- FSocketCommunicationServer (const std::string SendAddress="tcp://127.0.0.1:5555", const std::string RecvAddress="tcp://127.0.0.1:5556", const int PollTimeout=10)
- bool EventOccured ()

Notifies when an Event Has Occured In the Socket.
- bool SendArrayBuffer (const float ∗MessageData, size_t Size, ComSendFlags SendFlags=ComSendFlags↩ ::none)

  Sends an Array of Data over the Socket, using a Buffer.
- bool SendArrayBuffer (const float MessageData[ ], ComSendFlags SendFlags=ComSendFlags::none)

  Sends an Array of Data over the Socket, using a Buffer.
- bool SendArrayBuffer (const TArray< float > &ArrayMessage, ComSendFlags SendFlags=ComSendFlags↩ ::none)

  Sends an Array of Data over the Socket, using a Buffer.
- bool SendArrayMessage (const float ∗MessageData, size_t Size, ComSendFlags SendFlags=ComSend↩ Flags::none)

  Sends an Array of Data over the Socket, using a message.
- bool SendArrayMessage (const float MessageData[ ], ComSendFlags SendFlags=ComSendFlags::none)

  Sends an Array of Data over the Socket, using a message.
- bool SendArrayMessage (const TArray< float > &ArrayMessage, ComSendFlags SendFlags=ComSend↩ Flags::none)

  Sends an Array of Data over the Socket, using a message.
- bool SendArrayMessageWithMeta (const float ∗MessageData, size_t Size, const TSharedRef< FJsonObject > &Metadata, ComSendFlags SendFlags=ComSendFlags::none)

  Sends an Array of Data over the Socket, using a message.
- bool SendArrayMessageWithMeta (const float MessageData[ ], const TSharedRef< FJsonObject > &Meta-data, ComSendFlags SendFlags=ComSendFlags::none)

  Sends an Array of Data over the Socket, using a message.
- bool SendArrayMessageWithMeta (const TArray< float > &ArrayMessage, const TSharedRef< FJsonObject > &Metadata, ComSendFlags SendFlags=ComSendFlags::none)

  Sends an Array of Data over the Socket, using a message.
- bool SendStringBuffer (const std::string StringMessage, ComSendFlags SendFlags=ComSendFlags::none)

  Sends a String Buffer over the Socket.
- bool SendJsonBuffer (const std::string JsonMessage, ComSendFlags SendFlags=ComSendFlags::none)

  Sends a JSON Buffer over the Socket.
- template<typename T >
  bool RecvArray (TArray< T > &OutArrayData, size_t Size, ComRecvFlags RecvFlag=ComRecvFlags::none)

  Recives an Array of Data from the Socket.
- bool RecvString (FString &OutStringMessage, ComRecvFlags RecvFlag=ComRecvFlags::none)

  Recives String Data From the Socket.
- bool RecvJson (FString &OutJsonMessage, ComRecvFlags RecvFlag=ComRecvFlags::none)

  Recieves JSON Data From The Socket.
- bool RecvStringMultipart (TArray< FString > &OutMessages, ComRecvFlags RecvFlag=ComRecvFlags↩ ::none)

  Receives An Array of String Data From The Socket.
- bool RecvStringMultipartWithMeta (TArray< FString > &OutMessages, TSharedPtr< FJsonObject > &Out↩ Metadata, ComRecvFlags RecvFlag=ComRecvFlags::none)

  Receives An Array of String Data From The Socket, With JSON Metadata.

Protected Member Functions

- bool RecvMultipartWithMeta (std::vector< zmq::message_t > &OutMultipartMessages, TSharedPtr< FJsonObject > &OutMetadata, ComRecvFlags RecvFlags)

  Recieves a Multipart Message From The Socket, and a Metadata Object.
- bool SerializeJSON (const TSharedRef< FJsonObject > &InJsonObject, FString &OutJsonString)

  Serializes the JSON Object into a JSON String.
- bool DeserializeJSON (const FString &InJsonString, TSharedPtr< FJsonObject > &OutJsonObject)

  Deserializes the JSON String into a JSON Object.

Protected Attributes

- zmq::context_t ∗ Context

  The Context Used for the Socket Communication.
- zmq::socket_t ∗ SendSocket

  The Socket Used For Sending Data.
- zmq::socket_t ∗ RecvSocket

  The Socket Used For Receiving Data.
- zmq::poller_t< int > ∗ Poller

  The Poller used for Polling for Events on the Receiving Socket.
- std::string SendAddress
- std::string RecvAddress
- int PollTimeout

  The Time Taken By The Poller To Look For Events.

4.29.1 Detailed Description

Definition at line 22 of file SocketCommunicationServer.h.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 FSocketCommunicationServer()

```
FSocketCommunicationServer::FSocketCommunicationServer (
                const std::string SendAddress = "tcp://127.0.0.1:5555",
                const std::string RecvAddress = "tcp://127.0.0.1:5556",
                const int PollTimeout = 10 )
```

Definition at line 8 of file SocketCommunicationServer.cpp.

```
    : SendAddress(SendAddress), RecvAddress(RecvAddress), PollTimeout(PollTimeout)
{
    Context = new zmq::context_t(1);
    if (Context == nullptr)
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("Failed to create ZMQ context"));
        return;
    }

    SendSocket = new zmq::socket_t(*Context, ZMQ_PUSH);
    if (SendSocket == nullptr)
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("Failed to create ZMQ socket"));
        return;
    }

    RecvSocket = new zmq::socket_t(*Context, ZMQ_PULL);
    if (RecvSocket == nullptr)
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("Failed to create ZMQ socket"));
        return;
    }

    Poller = new zmq::poller_t<int>();
    if (Poller == nullptr)
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("Failed to create ZMQ poller"));
        return;
    }

    SendSocket->connect(SendAddress);
    RecvSocket->bind(RecvAddress);

    Poller->add(*RecvSocket, zmq::event_flags::pollin);
}
```

4.29.2.2 ∼FSocketCommunicationServer()

```
FSocketCommunicationServer::∼FSocketCommunicationServer ( )
```

Definition at line 45 of file SocketCommunicationServer.cpp.

```
{
    Poller->remove(*RecvSocket);
    delete Poller; Poller = nullptr;

    SendSocket->disconnect(SendAddress);
    SendSocket->close();
    delete SendSocket; SendSocket = nullptr;

    RecvSocket->unbind(RecvAddress);
    RecvSocket->close();
    delete RecvSocket; RecvSocket = nullptr;

    Context->shutdown();
    Context->close();
    delete Context; Context = nullptr;
}
```

4.29.3 Member Function Documentation

4.29.3.1 DeserializeJSON()

```
bool FSocketCommunicationServer::DeserializeJSON (
            const FString & InJsonString,
            TSharedPtr< FJsonObject > & OutJsonObject )  [protected]
```

Deserializes the JSON String into a JSON Object.

Parameters

InJsonString	- The JSON String To Deserialize.
OutJsonObject	- The Returned JSON Object from Deserialization.

Returns

True, if the JSON was Successfuly in Deserialization. Otherwise False.

Definition at line 444 of file SocketCommunicationServer.cpp.

```
{
    return FJsonSerializer::Deserialize(TJsonReaderFactory<TCHAR>::Create(InJsonString),
  OutJsonObject);
}
```

4.29.3.2 EventOccured()

```
bool FSocketCommunicationServer::EventOccured ( )
```

Notifies when an Event Has Occured In the Socket.

Returns

True, When an Event was Recived from the Socket. Otherwise False.

Definition at line 63 of file SocketCommunicationServer.cpp.

```
{
    std::vector<zmq::poller_event<int» PollEvents(1);
    if (Poller->wait_all(PollEvents, std::chrono::milliseconds(PollTimeout)) > 0)
    {
        PollEvents.clear();
        return true;
    }

    PollEvents.clear();
    return false;
}
```

4.29.3.3 RecvArray()

```
template<typename T >
bool FSocketCommunicationServer::RecvArray (
            TArray< T > & OutArrayData,
            size_t Size,
            ComRecvFlags RecvFlag = ComRecvFlags::none )
```

Recives an Array of Data from the Socket.

Template Parameters

T	The Type To Cast The Recived Data In The Array To.

Parameters

OutArrayData	- The Returned Array of Data From The Socket.
Size	- The Size of the Data Recieved From The Array.
RecvFlag	- The Recv Flags To Use When Recieving The Data.

Returns

True, if the Data was Recived from the Socket Successfully. False, if an error occurs in receiving.

Definition at line 303 of file SocketCommunicationServer.cpp.

```
{
    zmq::message_t RecvMessage;

    auto Result = RecvSocket->recv(RecvMessage, RecvFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Recv Array || Recv %d bytes"),
      Result.value());

        OutArrayData.Append(RecvMessage.data<T>(), Result.value());

        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Recv Array || EAGAIN Error
      Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.4 RecvJson()

```
bool FSocketCommunicationServer::RecvJson (
            FString & OutJsonMessage,
            ComRecvFlags RecvFlag = ComRecvFlags::none )
```

Recieves JSON Data From The Socket.

Parameters

OutJsonMessage	- Returns the JSON String Data Recived From the Socket.
RecvFlag	- The Recv Flags To Use When Recieving The Data.

Returns

True, if the Data was Recived from the Socket Successfully. False, if an error occurd in receiving.

Definition at line 348 of file SocketCommunicationServer.cpp.

```
{
    zmq::message_t RecvMessage;

    auto Result = RecvSocket->recv(RecvMessage, RecvFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Recv JSON || Recv %d bytes"),
   Result.value());

        OutJsonMessage = FString(Result.value(), UTF8_TO_TCHAR(RecvMessage.data()));

        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Recv JSON || EAGAIN Error
   Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.5 RecvMultipartWithMeta()

```
bool FSocketCommunicationServer::RecvMultipartWithMeta (
            std::vector< zmq::message_t > & OutMultipartMessages,
            TSharedPtr< FJsonObject > & OutMetadata,
            ComRecvFlags RecvFlags )  [protected]
```

Recieves a Multipart Message From The Socket, and a Metadata Object.

Parameters

OutMultipartMessages	- Returns the Array of Messages Contained in The Multipart.
OutMetadata	- Returns the Metadata JSON Object from the Multipart.
RecvFlags	- The Recv Flags To Use When Recieving The Data.

Returns

True, if the Multipart was Recieved Successfully. False, if an error occured in receiving.

Definition at line 409 of file SocketCommunicationServer.cpp.

```
{
    auto Result = zmq::recv_multipart(*RecvSocket, std::back_inserter(OutMultipartMessages),
  RecvFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Recv Multipart || Recv %d
  messages"), Result.value());

        // Pop Metadata Messages from the Front of Array.
        zmq::message_t MetadataMessage = MoveTempIfPossible(OutMultipartMessages[0]);
        OutMultipartMessages.erase(OutMultipartMessages.begin());

        if (DeserializeJSON(FString(UTF8_TO_TCHAR(MetadataMessage.data()), MetadataMessage.size()),
  OutMetadata))
        {
            return true;
        }
        else
        {
            UE_LOG(LogOpenAccessibilityCom, Error, TEXT("|| Com Server: Recv Multipart || Failed to
  deserialize metadata ||"));
            return false;
        }
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Recv Multipart || EAGAIN Error
  Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.6 RecvString()

```
bool FSocketCommunicationServer::RecvString (
            FString & OutStringMessage,
            ComRecvFlags RecvFlag = ComRecvFlags::none )
```

Recives String Data From the Socket.

Parameters

OutStringMessage	- Returns the String Data Recived From the Socket.
RecvFlag	- The Recv Flags To Use When Recieving The Data.

Returns

True, if the Data was Recived from the Socket Successfully. False, if an error occurs in receiving.

Definition at line 325 of file SocketCommunicationServer.cpp.

```
{
    zmq::message_t RecvMessage;

    auto Result = RecvSocket->recv(RecvMessage, RecvFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Recv String || Recv %d bytes"),
  Result.value());

        OutStringMessage = FString(Result.value(), UTF8_TO_TCHAR(RecvMessage.data()));

        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {

        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Recv String || EAGAIN Error
  Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.7 RecvStringMultipart()

```
bool FSocketCommunicationServer::RecvStringMultipart (
            TArray< FString > & OutMessages,
            ComRecvFlags RecvFlag = ComRecvFlags::none )
```

Receives An Array of String Data From The Socket.

Parameters

OutMessages	- Returns the Multipart of String Data Received From the Socket.
RecvFlag	- The Recv Flags To Use When Recieving The Data.

Returns

True, if the Data was Received from the Socket Successfully. False, if an error occured in receiving.

Definition at line 370 of file SocketCommunicationServer.cpp.

```
{
    std::vector<zmq::message_t> RecvMessages;

    auto Result = zmq::recv_multipart(*RecvSocket, std::back_inserter(RecvMessages), RecvFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Recv Multipart || Recv %d
  messages"), Result.value());

        for (auto& Message : RecvMessages)
        {
            OutMessages.Add(FString(Message.size(), UTF8_TO_TCHAR(Message.data())));
        }

        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Recv Multipart || EAGAIN Error
  Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.8 RecvStringMultipartWithMeta()

```
bool FSocketCommunicationServer::RecvStringMultipartWithMeta (
             TArray< FString > & OutMessages,
             TSharedPtr< FJsonObject > & OutMetadata,
             ComRecvFlags RecvFlag = ComRecvFlags::none )
```

Receives An Array of String Data From The Socket, With JSON Metadata.

Parameters

OutMessages	- Returns the Received Array of String Data.
OutMetadata	- Returns a JSON Object containing Metadata.
RecvFlag	- The Recv Flags To Use When Recieving The Data.

Returns

True, if the Multipart was Received Successfully. False, if an error occured in receiving.

Definition at line 395 of file SocketCommunicationServer.cpp.

```
{
    std::vector<zmq::message_t> RecvMessages;
    if (!RecvMultipartWithMeta(RecvMessages, OutMetadata, RecvFlag))
        return false;

    for (auto& Message : RecvMessages)
    {
        OutMessages.Add(FString(Message.size(), UTF8_TO_TCHAR(Message.data())));
    }

    return true;
}
```

4.29.3.9 SendArrayBuffer() [1/3]

```
bool FSocketCommunicationServer::SendArrayBuffer (
             const float * MessageData,
             size_t Size,
             ComSendFlags SendFlags = ComSendFlags::none )
```

Sends an Array of Data over the Socket, using a Buffer.

Parameters

MessageData	- The Array of Message Data To Send.
Size	- The Size of the Provided Data Array.
SendFlags	- The Send Flags for when sending over the socket.

Returns

True, if the Buffer was Sent Successfully. False, if an error occurs in sending.

Definition at line 76 of file SocketCommunicationServer.cpp.

```
{
    auto Result = SendSocket->send(zmq::const_buffer(MessageData, Size * sizeof(float)), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"), Result.value(), Size * sizeof(float));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.10 SendArrayBuffer() [2/3]

```
bool FSocketCommunicationServer::SendArrayBuffer (
            const float MessageData[],
            ComSendFlags SendFlags = ComSendFlags::none )
```

Sends an Array of Data over the Socket, using a Buffer.

Parameters

MessageData	- The Array of Message Data To Send.
SendFlags	- The Send Flags for when sending over the socket.

Returns

True, if the Buffer was Sent Successfully. False, if an error occurs in sending.

Definition at line 93 of file SocketCommunicationServer.cpp.

```
{
    auto Result = SendSocket->send(zmq::const_buffer(MessageData, sizeof MessageData), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"), Result.value(), int(sizeof MessageData));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.11 SendArrayBuffer() [3/3]

```
bool FSocketCommunicationServer::SendArrayBuffer (
            const TArray< float > & ArrayMessage,
            ComSendFlags SendFlags = ComSendFlags::none )
```

Sends an Array of Data over the Socket, using a Buffer.

Parameters

ArrayMessage	- The Array of Message Data To Send.
SendFlags	- The Send Flags for when sending over the socket.

Returns

True, if the Buffer was Sent Successfully. False, if an error occurs in sending.

Definition at line 110 of file SocketCommunicationServer.cpp.

```
{
    auto Result = SendSocket->send(zmq::const_buffer(ArrayMessage.GetData(), ArrayMessage.Num() *
  sizeof(float)), SendFlag);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"),
  Result.value(), int(ArrayMessage.Num() * sizeof(float)));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
  Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.12 SendArrayMessage() [1/3]

```
bool FSocketCommunicationServer::SendArrayMessage (
            const float * MessageData,
            size_t Size,
            ComSendFlags SendFlags = ComSendFlags::none )
```

Sends an Array of Data over the Socket, using a message.

Parameters

MessageData	- The Array of Data To Send.
Size	- The Size of the Data in the Array.
SendFlags	- The Send Flags for when sending over the socket.

Returns

True, if the Message was Sent Successfully. False, if an error occurs in sending.

Definition at line 127 of file SocketCommunicationServer.cpp.

```
{
    auto Result = SendSocket->send(zmq::message_t(MessageData, Size * sizeof(float)), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"),
  Result.value(), Size * sizeof(float));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
    Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.13 SendArrayMessage() [2/3]

```
bool FSocketCommunicationServer::SendArrayMessage (
            const float MessageData[],
            ComSendFlags SendFlags = ComSendFlags::none )
```

Sends an Array of Data over the Socket, using a message.

Parameters

MessageData	- The Array of Data To Send.
SendFlags	- The Send Flags To Use When Sending The Data.

Returns

True, if the Message was Sent Successfully. False, if an error occurs in sending.

Definition at line 144 of file SocketCommunicationServer.cpp.

```
{
    auto Result = SendSocket->send(zmq::message_t(MessageData, sizeof MessageData), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"),
    Result.value(), int(sizeof MessageData));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
    Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.14 SendArrayMessage() [3/3]

```
bool FSocketCommunicationServer::SendArrayMessage (
            const TArray< float > & ArrayMessage,
            ComSendFlags SendFlags = ComSendFlags::none )
```

Sends an Array of Data over the Socket, using a message.

Parameters

ArrayMessage	- The Array of Data To Send.
SendFlags	- The Send Flags To Use When Sending The Data.

Returns

True, if the Message was Sent Successfully. False, if an error occurs in sending.

Definition at line 161 of file SocketCommunicationServer.cpp.

```
{
    auto Result = SendSocket->send(zmq::message_t(ArrayMessage.GetData(), ArrayMessage.Num() *
  sizeof(float)), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"),
  Result.value(), int(ArrayMessage.Num() * sizeof(float)));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
  Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.15 SendArrayMessageWithMeta() [1/3]

```
bool FSocketCommunicationServer::SendArrayMessageWithMeta (
            const float * MessageData,
            size_t Size,
            const TSharedRef< FJsonObject > & Metadata,
            ComSendFlags SendFlags = ComSendFlags::none )
```

Sends an Array of Data over the Socket, using a message.

Parameters

MessageData	- The Array of Data To Send.
Size	- The Size of The Data Array.
Metadata	- The JSON Metadata to Send With The Message.
SendFlags	- The Send Flags To Use When Sending The Data.

Returns

True, if the Message was Sent Successfully. False, if an error occurs in sending.

Definition at line 178 of file SocketCommunicationServer.cpp.

```
{
    FString MetaDataString;
    if (!SerializeJSON(Metadata, MetaDataString))
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("|| Com Server: Sent Array || Failed to serialize
  metadata ||"));
        return false;
    }

    std::vector<zmq::message_t> Messages;
    Messages.push_back(zmq::message_t(*MetaDataString, MetaDataString.Len() * sizeof(TCHAR)));
    Messages.push_back(zmq::message_t(MessageData, Size * sizeof(float)));

    auto Result = zmq::send_multipart(*SendSocket, Messages, SendFlags);

    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"),
    Result.value(), Size * sizeof(float));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
    Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.16 SendArrayMessageWithMeta() [2/3]

```
bool FSocketCommunicationServer::SendArrayMessageWithMeta (
            const float MessageData[],
            const TSharedRef< FJsonObject > & Metadata,
            ComSendFlags SendFlags = ComSendFlags::none )
```

Sends an Array of Data over the Socket, using a message.

Parameters

MessageData	- The Array of Data To Send.
Metadata	- The JSON Metadata to Send With The Message.
SendFlags	- The Send Flags To Use When Sending The Data.

Returns

True, if the Message was Sent Successfully. False, if an error occurs in sending.

Definition at line 207 of file SocketCommunicationServer.cpp.

```
{
    FString MetaDataString;
    if (!SerializeJSON(Metadata, MetaDataString))
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("|| Com Server: Sent Array || Failed to serialize
    metadata ||"));
        return false;
    }

    std::vector<zmq::message_t> Messages;
    Messages.push_back(zmq::message_t(*MetaDataString, MetaDataString.Len() * sizeof(TCHAR)));
    Messages.push_back(zmq::message_t(MessageData, sizeof MessageData));

    auto Result = zmq::send_multipart(*SendSocket, Messages, SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"),
    Result.value(), int(sizeof MessageData));

        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
    Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.17 SendArrayMessageWithMeta() [3/3]

```
bool FSocketCommunicationServer::SendArrayMessageWithMeta (
            const TArray< float > & ArrayMessage,
            const TSharedRef< FJsonObject > & Metadata,
            ComSendFlags SendFlags = ComSendFlags::none )
```

Sends an Array of Data over the Socket, using a message.

Parameters

ArrayMessage	- The Array of Data To Send.
Metadata	- The JSON Metadata to Send With The Message.
SendFlags	- The Send Flags To Use When Sending The Data.

Returns

True, if the Message was Sent Successfully. False, if an error occurs in sending.

Definition at line 236 of file SocketCommunicationServer.cpp.

```
{
    FString MetaDataString;
    if (!SerializeJSON(Metadata, MetaDataString))
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("|| Com Server: Sent Array || Failed to serialize
  metadata ||"));
        return false;
    }

    std::vector<zmq::message_t> Messages;
    Messages.push_back(zmq::message_t(*MetaDataString, MetaDataString.Len() * sizeof(TCHAR)));
    Messages.push_back(zmq::message_t(ArrayMessage.GetData(), ArrayMessage.Num() * sizeof(float)));

    auto Result = zmq::send_multipart(*SendSocket, Messages, SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d
  Messages"), Result.value(), Messages.size());

        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
  Occured ||"));

        return true;
    }

    return false;
}
```

4.29.3.18 SendJsonBuffer()

```
bool FSocketCommunicationServer::SendJsonBuffer (
            const std::string JsonMessage,
            ComSendFlags SendFlags = ComSendFlags::none )
```

Sends a JSON Buffer over the Socket.

Parameters

JsonMessage	- The JSOn String Data To Send.
SendFlags	- The Send Flags To Use When Sending The Data.

Returns

True, if the Buffer was Sent Successfully. False, if an error occurs in sending.

Definition at line 283 of file SocketCommunicationServer.cpp.

```
{
    auto Result = SendSocket->send(zmq::const_buffer(JsonMessage.c_str(), JsonMessage.size()), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent JSON || Sent %d of %d bytes"), Result.value(), JsonMessage.size());
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent JSON || EAGAIN Error Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.19 SendStringBuffer()

```
bool FSocketCommunicationServer::SendStringBuffer (
            const std::string StringMessage,
            ComSendFlags SendFlags = ComSendFlags::none )
```

Sends a String Buffer over the Socket.

Parameters

StringMessage	- The String Data To Send.
SendFlags	- The Send Flags To Use When Sending The Data.

Returns

True, if the Buffer was Sent Successfully. False, if an error occurs in sending.

Definition at line 266 of file SocketCommunicationServer.cpp.

```
{
    auto Result = SendSocket->send(zmq::const_buffer(StringMessage.c_str(), StringMessage.size()), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent String || Sent %d of %d bytes"), Result.value(), StringMessage.size());
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent String || EAGAIN Error Occured ||"));
        return true;
    }

    return false;
}
```

4.29.3.20 SerializeJSON()

```
bool FSocketCommunicationServer::SerializeJSON (
            const TSharedRef< FJsonObject > & InJsonObject,
            FString & OutJsonString )  [protected]
```

Serializes the JSON Object into a JSON String.

Parameters

InJsonObject	- The JSON Object To Serialize.
OutJsonString	- The Returned Serialized JSON String from Serialization.

Returns

True, if the JSON Object was Successful in Serialization. Otherwise False.

Definition at line 439 of file SocketCommunicationServer.cpp.

```
{
    return FJsonSerializer::Serialize(InJsonObject,
  TJsonWriterFactory<TCHAR>::Create(&OutJsonString));
}
```

4.29.4 Member Data Documentation

4.29.4.1 Context

```
zmq::context_t* FSocketCommunicationServer::Context  [protected]
```

The Context Used for the Socket Communication.

Definition at line 205 of file SocketCommunicationServer.h.

4.29.4.2 Poller

```
zmq::poller_t<int>* FSocketCommunicationServer::Poller  [protected]
```

The Poller used for Polling for Events on the Receiving Socket.

Definition at line 220 of file SocketCommunicationServer.h.

4.29.4.3 PollTimeout

```
int FSocketCommunicationServer::PollTimeout  [protected]
```

The Time Taken By The Poller To Look For Events.

Definition at line 228 of file SocketCommunicationServer.h.

4.29.4.4 RecvAddress

```
std::string FSocketCommunicationServer::RecvAddress  [protected]
```

Definition at line 223 of file SocketCommunicationServer.h.

4.29.4.5 RecvSocket

```
zmq::socket_t* FSocketCommunicationServer::RecvSocket  [protected]
```

The Socket Used For Receiving Data.

Definition at line 215 of file SocketCommunicationServer.h.

4.29.4.6 SendAddress

```
std::string FSocketCommunicationServer::SendAddress  [protected]
```

Definition at line 222 of file SocketCommunicationServer.h.

4.29.4.7 SendSocket

```
zmq::socket_t* FSocketCommunicationServer::SendSocket  [protected]
```

The Socket Used For Sending Data.

Definition at line 210 of file SocketCommunicationServer.h.

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

- Source/OpenAccessibilityCommunication/Public/SocketCommunicationServer.h
- Source/OpenAccessibilityCommunication/Private/SocketCommunicationServer.cpp

4.30 FTranscriptionVisualizer Class Reference

Public Member Functions

- virtual bool Tick (float DeltaTime)
- void ConstructVisualizer ()

  Constructs the Visualizer Window, and Its Content.
- void UpdateVisualizer ()

  Updates the Visualizer Window, If Active.
- void ReparentWindow ()

  Reparents the Visualizer Window to the Active Window.
- void MoveVisualizer ()

  Moves the Visualizer Window to the Active Window Position.
- void OnTranscriptionRecieved (TArray< FString > InTranscription)

  Callback for when Transcriptions are Recieved From Transcribed Audio.

Protected Member Functions

- bool GetTopScreenVisualizerPosition (FVector2D &OutPosition)

  Gets the Position of the Visualizer for the Top Active Screen.
- bool GetDisplayVisualizerPosition (FVector2D &OutPosition)

  Gets the Position of the Visualizer for the Last Active Display.
- void RegisterTicker ()

  Registers the Ticker for the Visualizer.
- void UnregisterTicker ()

  Unregisters the Ticker for the Visualizer.

Protected Attributes

- FTSTicker::FDelegateHandle TickDelegateHandle
- TWeakPtr< SWindow > VisWindow

  The Visualizers Containing Window.
- TWeakPtr< class SAccessibilityTranscriptionVis > VisContent

  The Content of the Visualizer Window.

4.30.1 Detailed Description

Definition at line 7 of file TranscriptionVisualizer.h.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 FTranscriptionVisualizer()

```
FTranscriptionVisualizer::FTranscriptionVisualizer ( )
```

Definition at line 7 of file TranscriptionVisualizer.cpp.

```
{
    RegisterTicker();
}
```

4.30.2.2 ∼FTranscriptionVisualizer()

```
FTranscriptionVisualizer::~FTranscriptionVisualizer ( )
```

Definition at line 12 of file TranscriptionVisualizer.cpp.

```
{
    UnregisterTicker();
}
```

4.30.3 Member Function Documentation

4.30.3.1 ConstructVisualizer()

```
void FTranscriptionVisualizer::ConstructVisualizer ( )
```

Constructs the Visualizer Window, and Its Content.

Definition at line 31 of file TranscriptionVisualizer.cpp.

```
{
    TSharedPtr<SAccessibilityTranscriptionVis> MenuContent = SNew(SAccessibilityTranscriptionVis)
        .VisAmount(2);

    MenuContent->ForceVolatile(true);

    FDisplayMetrics DisplayMetrics;
    FSlateApplication::Get().GetDisplayMetrics(DisplayMetrics);

    FVector2D VisPosition = FVector2D();

    if (FSlateApplication::Get().GetActiveTopLevelRegularWindow().IsValid())
    {
        VisPosition =
  FSlateApplication::Get().GetActiveTopLevelRegularWindow()->GetPositionInScreen();
    }
    VisPosition.X = DisplayMetrics.PrimaryDisplayWidth;
    VisPosition.Y = DisplayMetrics.PrimaryDisplayHeight;

    TSharedRef<SWindow> MenuWindow = SNew(SWindow)
        .Type(EWindowType::Normal)
        .SizingRule(ESizingRule::Autosized)
        .ScreenPosition(VisPosition)
        .ClientSize(FVector2D(10, 10))
        .IsPopupWindow(true)
        //.InitialOpacity(0.5f)
        .SupportsTransparency(EWindowTransparency::PerWindow)
        .ActivationPolicy(EWindowActivationPolicy::Always)
        .AdjustInitialSizeAndPositionForDPIScale(true)
        [
            MenuContent.ToSharedRef()
        ];

    TSharedPtr<SWindow> TopLevelWindow = FSlateApplication::Get().GetActiveTopLevelRegularWindow();

    MenuWindow->AssignParentWidget(TopLevelWindow);
    FSlateApplication::Get().AddWindowAsNativeChild(MenuWindow , TopLevelWindow.ToSharedRef(), true);

    VisWindow = MenuWindow.ToWeakPtr();
    VisContent = MenuContent.ToWeakPtr();
}
```

4.30.3.2 GetDisplayVisualizerPosition()

```
bool FTranscriptionVisualizer::GetDisplayVisualizerPosition (
            FVector2D & OutPosition )  [protected]
```

Gets the Position of the Visualizer for the Last Active Display.

Parameters

OutPosition	

Definition at line 145 of file TranscriptionVisualizer.cpp.

```
{
    FDisplayMetrics DisplayMetrics;
    FSlateApplication::Get().GetDisplayMetrics(DisplayMetrics);

    OutPosition.X = DisplayMetrics.PrimaryDisplayWidth;
    OutPosition.Y = DisplayMetrics.PrimaryDisplayHeight;

    return true;
}
```

4.30.3.3 GetTopScreenVisualizerPosition()

```
bool FTranscriptionVisualizer::GetTopScreenVisualizerPosition (
            FVector2D & OutPosition )  [protected]
```

Gets the Position of the Visualizer for the Top Active Screen.

Parameters

OutPosition	

Definition at line 128 of file TranscriptionVisualizer.cpp.

```
{
    TSharedPtr<SWindow> TopLevelWindow = FSlateApplication::Get().GetActiveTopLevelRegularWindow();
    if (!TopLevelWindow.IsValid())
        return false;

    FVector2D ActiveWindowPosition = TopLevelWindow->GetPositionInScreen();
    FVector2D ActiveWindowBounds = TopLevelWindow->GetClientSizeInScreen();

    TSharedPtr<SWindow> VisWindowPtr = VisWindow.Pin();

    OutPosition.X = (ActiveWindowPosition.X + ActiveWindowBounds.X / 2) -
      (VisWindowPtr->GetClientSizeInScreen().X / 2);
    OutPosition.Y = (ActiveWindowPosition.Y + ActiveWindowBounds.Y - 50) -
      VisWindowPtr->GetClientSizeInScreen().Y;

    return true;
}
```

4.30.3.4 MoveVisualizer()

```
void FTranscriptionVisualizer::MoveVisualizer ( )
```

Moves the Visualizer Window to the Active Window Position.

Definition at line 108 of file TranscriptionVisualizer.cpp.

```
{
    FVector2D NewPosition = FVector2D();

    if (!GetTopScreenVisualizerPosition(NewPosition))
    {
        GetDisplayVisualizerPosition(NewPosition);
    }

    VisWindow.Pin()->MoveWindowTo(NewPosition);
}
```

4.30.3.5 OnTranscriptionRecieved()

```
void FTranscriptionVisualizer::OnTranscriptionRecieved (
            TArray< FString > InTranscription )
```

Callback for when Transcriptions are Recieved From Transcribed Audio.

Parameters

InTranscription	Incoming Array of Transcription Strings.

Definition at line 120 of file TranscriptionVisualizer.cpp.

```
{
    for (int i = 0; i < InTranscription.Num(); i++)
    {
        VisContent.Pin()->UpdateTopTranscription(InTranscription[i]);
    }
}
```

4.30.3.6 RegisterTicker()

```
void FTranscriptionVisualizer::RegisterTicker ( )  [protected]
```

Registers the Ticker for the Visualizer.

Definition at line 156 of file TranscriptionVisualizer.cpp.

```
{
    FTickerDelegate TickDelegate = FTickerDelegate::CreateRaw(this, &FTranscriptionVisualizer::Tick);

    TickDelegateHandle = FTSTicker::GetCoreTicker().AddTicker(TickDelegate);
}
```

4.30.3.7 ReparentWindow()

```
void FTranscriptionVisualizer::ReparentWindow ( )
```

Reparents the Visualizer Window to the Active Window.

Definition at line 86 of file TranscriptionVisualizer.cpp.

```
{
    TSharedPtr<SWindow> TopLevelActiveWindow =
  FSlateApplication::Get().GetActiveTopLevelRegularWindow();
    if (!TopLevelActiveWindow.IsValid())
        return;

    TSharedPtr<SWindow> VisWindowPtr = VisWindow.Pin();

    if (TopLevelActiveWindow == VisWindow.Pin() ||
        TopLevelActiveWindow->GetContent() == VisWindowPtr->GetParentWidget())
        return;

    TSharedPtr<SWindow> PrevParentWindow = VisWindowPtr->GetParentWindow();
    if (PrevParentWindow.IsValid())
    {
        PrevParentWindow->RemoveDescendantWindow(VisWindowPtr.ToSharedRef());
    }

    VisWindowPtr->AssignParentWidget(TopLevelActiveWindow);
    TopLevelActiveWindow->AddChildWindow(VisWindowPtr.ToSharedRef());
}
```

4.30.3.8 Tick()

```
bool FTranscriptionVisualizer::Tick (
              float DeltaTime )   [virtual]
```

Definition at line 17 of file TranscriptionVisualizer.cpp.

```
{
    if (VisWindow.IsValid())
    {
        UpdateVisualizer();
    }
    else if (FSlateApplication::Get().GetActiveTopLevelRegularWindow().IsValid() &&
  FSlateApplication::Get().IsActive())
    {
        ConstructVisualizer();
    }

    return true;
}
```

4.30.3.9 UnregisterTicker()

```
void FTranscriptionVisualizer::UnregisterTicker ( )   [protected]
```

Unregisters the Ticker for the Visualizer.

Definition at line 163 of file TranscriptionVisualizer.cpp.

```
{
    FTSTicker::GetCoreTicker().RemoveTicker(TickDelegateHandle);
}
```

4.30.3.10 UpdateVisualizer()

```
void FTranscriptionVisualizer::UpdateVisualizer ( )
```

Updates the Visualizer Window, If Active.

Definition at line 73 of file TranscriptionVisualizer.cpp.

```
{
    if (FSlateApplication::Get().IsActive())
    {
        VisWindow.Pin()->ShowWindow();

        // ReparentWindow();

        MoveVisualizer();
    }
    else VisWindow.Pin()->HideWindow();
}
```

4.30.4 Member Data Documentation

4.30.4.1 TickDelegateHandle

```
FTSTicker::FDelegateHandle FTranscriptionVisualizer::TickDelegateHandle  [protected]
```

Definition at line 75 of file TranscriptionVisualizer.h.

4.30.4.2 VisContent

```
TWeakPtr<class SAccessibilityTranscriptionVis> FTranscriptionVisualizer::VisContent  [protected]
```

The Content of the Visualizer Window.

Definition at line 87 of file TranscriptionVisualizer.h.

4.30.4.3 VisWindow

```
TWeakPtr<SWindow> FTranscriptionVisualizer::VisWindow  [protected]
```

The Visualizers Containing Window.

Definition at line 82 of file TranscriptionVisualizer.h.

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

- Source/OpenAccessibility/Public/TranscriptionVisualizer.h
- Source/OpenAccessibility/Private/TranscriptionVisualizer.cpp

4.31 UAccessibilityGraphEditorContext::FTreeViewTickRequirements Struct Reference

Public Attributes

- FString PrevSearchText
- int32 PrevNumItemsBeingObserved
- int32 PrevNumGeneratedChildren
- double PrevScrollDistance

4.31.1 Detailed Description

Definition at line 146 of file AccessibilityGraphEditorContext.h.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 FTreeViewTickRequirements()

`UAccessibilityGraphEditorContext::FTreeViewTickRequirements::FTreeViewTickRequirements ( ) [inline]`

Definition at line 150 of file AccessibilityGraphEditorContext.h.

```
            : PrevSearchText(FString())
            , PrevNumItemsBeingObserved(-1)
            , PrevNumGeneratedChildren(-1)
            , PrevScrollDistance(-1.00)
        { }
```

4.31.3 Member Data Documentation

4.31.3.1 PrevNumGeneratedChildren

`int32 UAccessibilityGraphEditorContext::FTreeViewTickRequirements::PrevNumGeneratedChildren`

Definition at line 159 of file AccessibilityGraphEditorContext.h.

4.31.3.2 PrevNumItemsBeingObserved

`int32 UAccessibilityGraphEditorContext::FTreeViewTickRequirements::PrevNumItemsBeingObserved`

Definition at line 158 of file AccessibilityGraphEditorContext.h.

4.31.3.3 PrevScrollDistance

```
double UAccessibilityGraphEditorContext::FTreeViewTickRequirements::PrevScrollDistance
```

Definition at line 160 of file AccessibilityGraphEditorContext.h.

4.31.3.4 PrevSearchText

```
FString UAccessibilityGraphEditorContext::FTreeViewTickRequirements::PrevSearchText
```

Definition at line 157 of file AccessibilityGraphEditorContext.h.

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

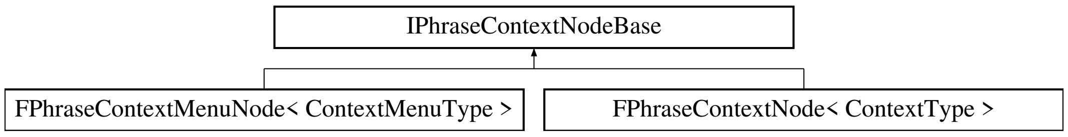
- Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityGraphEditorContext.h

4.32 IPhraseContextNodeBase Class Reference

Base Abstract Class For Phrase Context Nodes, that are required to have a Context Node.

```
#include <IPhraseContextNode.h>
```

Inheritance diagram for IPhraseContextNodeBase:


IPhraseContextNodeBase
FPhraseContextMenuNode< ContextMenuType >
FPhraseContextNode< ContextType >

Protected Member Functions

- virtual bool HasContextObject (TArray< UPhraseTreeContextObject ∗ > InContextObjects) const =0

  Checks if the Given Context Array Contains Context Objects.
- virtual UPhraseTreeContextObject ∗ CreateContextObject (FParseRecord &Record)=0

  Creates a Context Object, using Record Inputs.
- virtual void ConstructContextChildren (TArray< TSharedPtr< class FPhraseNode > > &InChildNodes)=0

  Constructs the Context Nodes Children, from Given Child Nodes. Allowing for Inclusion of Utility Nodes in relation to the Context.

4.32.1 Detailed Description

Base Abstract Class For Phrase Context Nodes, that are required to have a Context Node.

Definition at line 12 of file IPhraseContextNode.h.

4.32.2 Member Function Documentation

4.32.2.1 ConstructContextChildren()

```
virtual void IPhraseContextNodeBase::ConstructContextChildren (
            TArray< TSharedPtr< class FPhraseNode > > & InChildNodes )  [protected], [pure
virtual]
```

Constructs the Context Nodes Children, from Given Child Nodes. Allowing for Inclusion of Utility Nodes in relation to the Context.

Parameters

InChildNodes	- An Array of the Nodes Children.

4.32.2.2 CreateContextObject()

```
virtual UPhraseTreeContextObject * IPhraseContextNodeBase::CreateContextObject (
            FParseRecord & Record )  [protected], [pure virtual]
```

Creates a Context Object, using Record Inputs.

Returns

The Created Context Object, otherwise nullptr

Implemented in FPhraseContextNode< ContextType >, and FPhraseContextMenuNode< ContextMenuType >.

4.32.2.3 HasContextObject()

```
virtual bool IPhraseContextNodeBase::HasContextObject (
            TArray< UPhraseTreeContextObject * > InContextObjects ) const  [protected], [pure
virtual]
```

Checks if the Given Context Array Contains Context Objects.

Parameters

InContextObjects	- The Array To Check For Context Objects.

Returns

True, if their is Context Objects in the Given Array.

Implemented in FPhraseContextNode< ContextType >, and FPhraseContextMenuNode< ContextMenuType >.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/IPhraseContextNode.h

4.33 IPhraseNodeBase Class Reference

Public Member Functions

- virtual bool IsLeafNode () const =0

  States if the Phrase Node is a LeafNode.
- virtual bool HasLeafChild () const =0

  States if the the Single Child Node is a Leaf Node, if it exists.
- virtual bool RequiresPhrase (const FString InPhrase)=0

  Checks if the Given Phrase is Bound to the Node.
- virtual FParseResult ParsePhrase (TArray< FString > &InPhraseWordArray, FParseRecord &InParse↩ Record)=0

  Parses the phrase down the given Node, propagating down child nodes if required.
- virtual FParseResult ParsePhraseAsContext (TArray< FString > &InPhraseWordArray, FParseRecord &In↩ ParseRecord)=0

  Parses the phrase down the given node, propagating down child nodes if required. Missed Pop of the Phrase Array from this Node.

4.33.1 Detailed Description

Definition at line 10 of file PhraseNode.h.

4.33.2 Member Function Documentation

4.33.2.1 HasLeafChild()

```
virtual bool IPhraseNodeBase::HasLeafChild ( ) const  [pure virtual]
```

States if the the Single Child Node is a Leaf Node, if it exists.

Returns

4.33.2.2 IsLeafNode()

```
virtual bool IPhraseNodeBase::IsLeafNode ( ) const  [pure virtual]
```

States if the Phrase Node is a LeafNode.

Returns

true, if the Node is a Leaf Node otherwise false.

4.33.2.3 ParsePhrase()

```
virtual FParseResult IPhraseNodeBase::ParsePhrase (
            TArray< FString > & InPhraseWordArray,
            FParseRecord & InParseRecord )  [pure virtual]
```

Parses the phrase down the given Node, propagating down child nodes if required.

Parameters

InPhraseWordArray	The Array of Phrase Strings to Propogate against.
InParseRecord	The Record of Propagation of collected context's and inputs.

Returns

Returns the Result of the propogation, including any key nodes met.

4.33.2.4 ParsePhraseAsContext()

```
virtual FParseResult IPhraseNodeBase::ParsePhraseAsContext (
            TArray< FString > & InPhraseWordArray,
            FParseRecord & InParseRecord )  [pure virtual]
```

Parses the phrase down the given node, propagating down child nodes if required. Missed Pop of the Phrase Array from this Node.

Parameters

InPhraseWordArray	
InParseRecord	

Returns

Returns the Result of the propogation, including any key nodes met.

4.33.2.5 RequiresPhrase()

```
virtual bool IPhraseNodeBase::RequiresPhrase (
             const FString InPhrase )  [pure virtual]
```

Checks if the Given Phrase is Bound to the Node.

Parameters

InPhrase	The Phrase String to Compare Against.

Returns

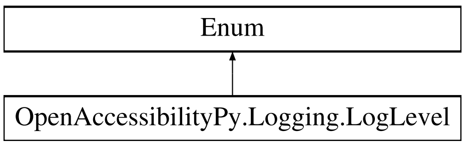
True, if the Node requires the given phrase string otherwise false.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseNode.h

4.34 OpenAccessibilityPy.Logging.LogLevel Class Reference

Inheritance diagram for OpenAccessibilityPy.Logging.LogLevel:



Static Public Attributes

- int INFO = 0
- int WARNING = 1
- int ERROR = 2

4.34.1 Detailed Description

Definition at line 4 of file Logging.py.

4.34.2 Member Data Documentation

4.34.2.1 ERROR

```
int OpenAccessibilityPy.Logging.LogLevel.ERROR = 2  [static]
```

Definition at line 7 of file Logging.py.

4.34.2.2 INFO

```
int OpenAccessibilityPy.Logging.LogLevel.INFO = 0  [static]
```

Definition at line 5 of file Logging.py.

4.34.2.3 WARNING

```
int OpenAccessibilityPy.Logging.LogLevel.WARNING = 1  [static]
```

Definition at line 6 of file Logging.py.

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

- Content/Python/OpenAccessibilityPy/Logging.py

4.35 TestWhisper.ModelInfo Class Reference

4.35.1 Detailed Description

Definition at line 7 of file TestWhisper.py.

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

- Content/Python/TestWhisper.py

4.36 NumericParser Class Reference

Static Public Member Functions

- static bool IsValidNumeric (const FString &StringToCheck, bool ConvertToUpper=true)

  Checks if the String is a Valid Numeric in Comparison to its String Permutations.
- static void StringToNumeric (FString &NumericString, bool ConvertToUpper=true)

  Converts a String to its Numeric Permutation.

4.36.1 Detailed Description

Definition at line 7 of file Utils.h.

4.36.2 Member Function Documentation

4.36.2.1 IsValidNumeric()

```
bool NumericParser::IsValidNumeric (
            const FString & StringToCheck,
            bool ConvertToUpper = true )  [static]
```

Checks if the String is a Valid Numeric in Comparison to its String Permutations.

Parameters

StringToCheck	- The String To Check if it is a Numeric.
ConvertToUpper	- Should The String Be Converted To Upper before Comparison.

Returns

Definition at line 7 of file Utils.cpp.

```
00008 {
00009     return StringMappings.Contains(ConvertToUpper ? StringToCheck.ToUpper() : StringToCheck);
00010 }
```

4.36.2.2 StringToNumeric()

```
void NumericParser::StringToNumeric (
            FString & NumericString,
            bool ConvertToUpper = true )  [static]
```

Converts a String to its Numeric Permutation.

Parameters

NumericString	- The String To Convert To Numeric.
ConvertToUpper	- Should The String Be Converted To Upper before Conversion.

Definition at line 12 of file Utils.cpp.

```
00013 {
00014     if (const FString* FoundMapping = StringMappings.Find(NumericString))
00015     {
00016         NumericString = ConvertToUpper ? *FoundMapping->ToUpper() : *FoundMapping;
```

```
    }
    else UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Numeric Parser || No Mapping Found for
  String: %s ||"), *NumericString);
}
```

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/Utils.h
- Source/OpenAccessibilityCommunication/Private/PhraseTree/Utils.cpp

4.37 OAEditorAccessibilityManager Class Reference

4.37.1 Detailed Description

Definition at line 10 of file OAEditorAccessibilityManager.h.

4.37.2 Constructor & Destructor Documentation

4.37.2.1 OAEditorAccessibilityManager()

```
OAEditorAccessibilityManager::OAEditorAccessibilityManager ( )
```

Definition at line 6 of file OAEditorAccessibilityManager.cpp.

```
{
}
```

4.37.2.2 ∼OAEditorAccessibilityManager()

```
OAEditorAccessibilityManager::∼OAEditorAccessibilityManager ( )
```

Definition at line 10 of file OAEditorAccessibilityManager.cpp.

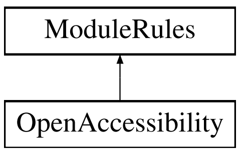
```
{
}
```

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

- Source/OpenAccessibility/Public/OAEditorAccessibilityManager.h
- Source/OpenAccessibility/Private/OAEditorAccessibilityManager.cpp

4.38 OpenAccessibility Class Reference

Inheritance diagram for OpenAccessibility:



Public Member Functions

- OpenAccessibility (ReadOnlyTargetRules Target)

4.38.1 Detailed Description

Definition at line 6 of file OpenAccessibility.Build.cs.

4.38.2 Constructor & Destructor Documentation

4.38.2.1 OpenAccessibility()

```
OpenAccessibility.OpenAccessibility (
              ReadOnlyTargetRules Target )   [inline]
```

Definition at line 8 of file OpenAccessibility.Build.cs.

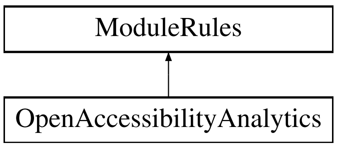
```
00008                                                             : base(Target)
00009     {
00010         PCHUsage = ModuleRules.PCHUsageMode.UseExplicitOrSharedPCHs;
00011 
00012         PublicIncludePaths.AddRange(
00013             new string[] {
00014                 // ... add public include paths required here ...
00015             }
00016             );
00017 
00018         PrivateIncludePaths.AddRange(
00019             new string[] {
00020                 // ... add other private include paths required here ...
00021             }
00022             );
00023 
00024 
00025         PublicDependencyModuleNames.AddRange(
00026             new string[]
00027             {
00028                 "Core",
00029                 // ... add other public dependencies that you statically link with here ...
00030             }
00031             );
00032 
00033 
00034         PrivateDependencyModuleNames.AddRange(
00035             new string[]
00036             {
00037                 // Internal Plugin Modules
00038                 "OpenAccessibilityCommunication",
00039 
00040                 // Core Modules
00041                 "CoreUObject",
00042                 "Engine",
00043 
00044                 // Editor Modules
00045                 "UnrealEd",
00046                 "GraphEditor",
00047                 "Kismet",
00048                 "AIModule",
00049 
00050                 // Slate UI
00051                 "Slate",
00052                 "SlateCore",
00053                 "EditorStyle",
00054             }
00055             );
00056 
00057 
00058         DynamicallyLoadedModuleNames.AddRange(
00059             new string[]
00060               {
00061                   // ... add any modules that your module loads dynamically here ...
00062               }
00063               );
00064 
00065           CircularlyReferencedDependentModules.AddRange(
00066               new string[]
00067               {
00068 
00069               }
00070           );
00071       }
```

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

- Source/OpenAccessibility/OpenAccessibility.Build.cs

4.39 OpenAccessibilityAnalytics Class Reference

Inheritance diagram for OpenAccessibilityAnalytics:



Public Member Functions

- OpenAccessibilityAnalytics (ReadOnlyTargetRules Target)

4.39.1 Detailed Description

Definition at line 6 of file OpenAccessibilityAnalytics.Build.cs.

4.39.2 Constructor & Destructor Documentation

4.39.2.1 OpenAccessibilityAnalytics()

```
OpenAccessibilityAnalytics.OpenAccessibilityAnalytics (
              ReadOnlyTargetRules Target )  [inline]
```

Definition at line 8 of file OpenAccessibilityAnalytics.Build.cs.

```
00008                                                             : base(Target)
00009       {
00010           PCHUsage = ModuleRules.PCHUsageMode.UseExplicitOrSharedPCHs;
00011 
00012           PublicIncludePaths.AddRange(
00013               new string[] {
00014                   // ... add public include paths required here ...
00015               }
00016               );
```

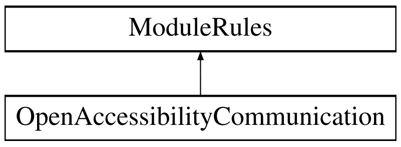
```

        PrivateIncludePaths.AddRange(
            new string[] {
                // ... add other private include paths required here ...
            }
            );


        PublicDependencyModuleNames.AddRange(
            new string[]
            {
                "Core",
                // ... add other public dependencies that you statically link with here ...
            }
            );


        PrivateDependencyModuleNames.AddRange(
            new string[]
            {
                "Engine",
            }
            );


        DynamicallyLoadedModuleNames.AddRange(
            new string[]
            {
                // ... add any modules that your module loads dynamically here ...
            }
            );

        CircularlyReferencedDependentModules.AddRange(
            new string[]
            {

            }
        );
    }
```

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

- Source/OpenAccessibilityAnalytics/OpenAccessibilityAnalytics.Build.cs

4.40 OpenAccessibilityCommunication Class Reference

Inheritance diagram for OpenAccessibilityCommunication:



Public Member Functions

- OpenAccessibilityCommunication (ReadOnlyTargetRules Target)

4.40.1 Detailed Description

Definition at line 7 of file OpenAccessibilityCommunication.Build.cs.

4.40.2 Constructor & Destructor Documentation

4.40.2.1 OpenAccessibilityCommunication()

```
OpenAccessibilityCommunication.OpenAccessibilityCommunication (
              ReadOnlyTargetRules Target )  [inline]
```

Definition at line 9 of file OpenAccessibilityCommunication.Build.cs.

```
                                                                       : base(Target)
        {
            PCHUsage = ModuleRules.PCHUsageMode.UseExplicitOrSharedPCHs;

            PublicIncludePaths.AddRange(
                new string[] {
                    // ... add public include paths required here ...
                }
                );

            PrivateIncludePaths.AddRange(
                new string[] {
                    // ... add other private include paths required here ...
                }
                );


            PublicDependencyModuleNames.AddRange(
                new string[]
                {
                    "Core",
                    // ... add other public dependencies that you statically link with here ...
                }
                );

            PrivateDependencyModuleNames.AddRange(
                new string[]
                {
                    // Internal Plugin Dependencies
                    "OpenAccessibilityAnalytics",

                    // Internal ThirdParty Dependencies
                    "ZeroMQ",

                    // Core Modules
                    "CoreUObject",
                    "Engine",
                    "Json",

                    // Editor Modules
                    "UnrealEd",
                    "Projects",

                    // Slate UI Modules
                    "Slate",
                    "SlateCore",

                    // Audio Modules
                    "AudioMixer",
                    "AudioCaptureCore",
                    "AudioCapture",
                    "InputCore",
                }
                );


            DynamicallyLoadedModuleNames.AddRange(
                new string[]
                {
                    // ... add any modules that your module loads dynamically here ...
                }
                );

            CircularlyReferencedDependentModules.AddRange(
                new string[]
                {

                }
            );
        }
```

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

- Source/OpenAccessibilityCommunication/OpenAccessibilityCommunication.Build.cs

4.41 OpenAccessibilityPy.OpenAccessibilityPy Class Reference

Public Member Functions

- def __init__ (self, int worker_count=2, str whisper_model="distil-small.en", str device="auto", str compute_↩ type="default", int poll_timeout=10)
- def __del__ (self)
- def Tick (self, float delta_time)
- def HandleTranscriptionRequest (self, np.ndarray recv_message, dict metadata=None)
- def Shutdown (self)

Public Attributes

- worker_pool
- whisper_interface
- com_server
- audio_resampler
- tick_handle
- pyshutdown_handle

4.41.1 Detailed Description

```
Python Runtime Class for Open Accessbility Plugin
```

Definition at line 34 of file __init__.py.

4.41.2 Constructor & Destructor Documentation

4.41.2.1 __init__()

```
def OpenAccessibilityPy.OpenAccessibilityPy.__init__ (
              self,
             int  worker_count = 2,
             str  whisper_model = "distil-small.en",
             str  device = "auto",
             str  compute_type = "default",
             int  poll_timeout = 10 )
```

```
Constructor of Python Runtime Class for Open Accessibility Plugin

Args:
    worker_count (int, optional): Amount of Thread Workers for Audio Transcription. Defaults to 2.
    whisper_model (str, optional): Hugging-Face Model Specifier for CTranslate Compatible Models. Defaults to
    device (str, optional): Device host for the Whisper Model (Can be "auto", "cpu", "cuda"). Defaults to "aut
    compute_type (str, optional): Data Structure for Whisper Compute. Defaults to "default".
    poll_timeout (int, optional): Amount of time (ms) for event polling on the Transcription Socket. Defaults
```

Definition at line 37 of file __init__.py.
```
    ):
        """Constructor of Python Runtime Class for Open Accessibility Plugin

        Args:
            worker_count (int, optional): Amount of Thread Workers for Audio Transcription. Defaults
    to 2.
            whisper_model (str, optional): Hugging-Face Model Specifier for CTranslate Compatible
    Models. Defaults to "distil-small.en".
            device (str, optional): Device host for the Whisper Model (Can be "auto", "cpu", "cuda").
    Defaults to "auto".
            compute_type (str, optional): Data Structure for Whisper Compute. Defaults to "default".
            poll_timeout (int, optional): Amount of time (ms) for event polling on the Transcription
    Socket. Defaults to 10.
        """

        self.worker_pool = ThreadPool(
            max_workers=worker_count, thread_name_prefix="TranscriptionWorker"
        )

        self.whisper_interface = WhisperInterface(
            model_name=whisper_model,
            device=device,
            compute_type=compute_type,
            transcribe_workers=worker_count,
        )
        self.com_server = CommunicationServer(
            send_socket_type=zmq.PUSH,
            recv_socket_type=zmq.PULL,
            poll_timeout=poll_timeout,
        )
        self.audio_resampler = AudioResampler(target_sample_rate=16000)

        self.tick_handle = ue.register_slate_post_tick_callback(self.Tick)

        self.pyshutdown_handle = ue.register_python_shutdown_callback(self.Shutdown)

```

4.41.2.2 __del__()

```
def OpenAccessibilityPy.OpenAccessibilityPy.__del__ (
                self )
```

Destructor of Python Runtime Class for Open Accessibility Plugin

Definition at line 79 of file __init__.py.
```
    def __del__(self):
        """Destructor of Python Runtime Class for Open Accessibility Plugin"""

        self.Shutdown()

```

4.41.3 Member Function Documentation

4.41.3.1 HandleTranscriptionRequest()

```
def OpenAccessibilityPy.OpenAccessibilityPy.HandleTranscriptionRequest (
                self,
                np.ndarray recv_message,
                dict metadata = None )
```

```
Handles Incoming Transcription Requests

Takes the Incoming AudioBuffer, Resamples it to 16kHz and Transcribes it using Whisper.

Args:
    recv_message (np.ndarray): ndarray of the incoming audio buffer.
    metadata (dict, optional): metadata of the incoming audio buffer, if any is recieved. Defaults to None.
```

Definition at line 100 of file __init__.py.

```
    ):
        """Handles Incoming Transcription Requests

        Takes the Incoming AudioBuffer, Resamples it to 16kHz and Transcribes it using Whisper.

        Args:
            recv_message (np.ndarray): ndarray of the incoming audio buffer.
            metadata (dict, optional): metadata of the incoming audio buffer, if any is recieved.
 Defaults to None.
        """

        Log(
            f"Handling Transcription Request | Message: {recv_message} | Size: {recv_message.size} |
 Shape: {recv_message.shape}"
        )

        sample_rate = metadata.get("sample_rate", 48000)
        num_channels = metadata.get("num_channels", 2)

        message_ndarray = self.audio_resampler.resample(
            recv_message, sample_rate, num_channels
        )

        trans_segments, trans_metadata = self.whisper_interface.process_audio_buffer(
            message_ndarray
        )

        encoded_segments = [
            transcription.text.encode() for transcription in trans_segments
        ]

        Log(f"Encoded Segments: {encoded_segments}")

        if len(encoded_segments) > 0:
            try:
                self.com_server.SendMultipartWithMeta(
                    message=encoded_segments, meta=trans_metadata
                )

            except:
                Log("Error Sending Encoded Transcription Segments", LogLevel.ERROR)

        else:
            Log("No Transcription Segments Returned", LogLevel.WARNING)

```

4.41.3.2 Shutdown()

```
def OpenAccessibilityPy.OpenAccessibilityPy.Shutdown (
                self )
```

```
Shutsdown the Python Runtime Components, and Forces a Garbage Collection.
```

Definition at line 145 of file __init__.py.

```
    def Shutdown(self):
        """Shutsdown the Python Runtime Components, and Forces a Garbage Collection."""

        if self.tick_handle:
            ue.unregister_slate_post_tick_callback(self.tick_handle)
            del self.tick_handle

        if self.worker_pool:
            self.worker_pool.shutdown(wait=False, cancel_futures=True)
            del self.worker_pool

        if self.audio_resampler:
            del self.audio_resampler

        if self.com_server:
            del self.com_server

        if self.whisper_interface:
            del self.whisper_interface

        # Force a Garbage Collection
        gc_collect()
```

4.41.3.3 Tick()

```
def OpenAccessibilityPy.OpenAccessibilityPy.Tick (
                self,
               float delta_time )
```

```
Tick Callback for Unreal Engine Slate Post Tick.

Detecting Incoming Transcription Requests and Handling them, through the Worker Pool.

Args:
    delta_time (float): Time since last tick
```

Definition at line 84 of file __init__.py.

```
00084     def Tick(self, delta_time: float):
00085         """Tick Callback for Unreal Engine Slate Post Tick.
00086 
00087         Detecting Incoming Transcription Requests and Handling them, through the Worker Pool.
00088 
00089         Args:
00090             delta_time (float): Time since last tick
00091         """
00092 
00093         if self.com_server.EventOccured():
00094             Log("Event Occured")
00095 
00096             message, metadata = self.com_server.ReceiveNDArrayWithMeta(dtype=np.float32)
00097 
00098             self.worker_pool.submit(self.HandleTranscriptionRequest, message, metadata)
00099 
```

4.41.4 Member Data Documentation

4.41.4.1 audio_resampler

```
OpenAccessibilityPy.OpenAccessibilityPy.audio_resampler
```

Definition at line 73 of file __init__.py.

4.41.4.2 com_server

```
OpenAccessibilityPy.OpenAccessibilityPy.com_server
```

Definition at line 68 of file __init__.py.

4.41.4.3 pyshutdown_handle

```
OpenAccessibilityPy.OpenAccessibilityPy.pyshutdown_handle
```

Definition at line 77 of file __init__.py.

4.41.4.4 tick_handle

```
OpenAccessibilityPy.OpenAccessibilityPy.tick_handle
```

Definition at line 75 of file __init__.py.

4.41.4.5 whisper_interface

```
OpenAccessibilityPy.OpenAccessibilityPy.whisper_interface
```

Definition at line 62 of file __init__.py.

4.41.4.6 worker_pool

```
OpenAccessibilityPy.OpenAccessibilityPy.worker_pool
```

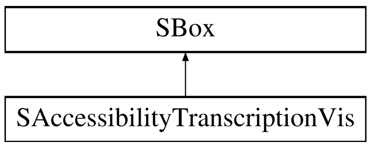
Definition at line 58 of file __init__.py.

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

- Content/Python/OpenAccessibilityPy/__init__.py

4.42 SAccessibilityTranscriptionVis Class Reference

Inheritance diagram for SAccessibilityTranscriptionVis:



Public Member Functions

- SLATE_BEGIN_ARGS (SAccessibilityTranscriptionVis)
- void Construct (const FArguments &InArgs)
- virtual void Tick (const FGeometry &AllottedGeometry, const double InCurrentTime, const float InDeltaTime) override
- void UpdateTopTranscription (const FString &InTopTranscription)

    Updates the Top Transcription Text, shifting all current transcriptions down.

Protected Attributes

- TWeakPtr< SVerticalBox > TranscriptionContainer

    The Container of the Transcription Slots.
- TArray< TWeakPtr< STextBlock > > TranscriptionSlots

    Array of the created Transcription Slots, displaying recieved transcriptions.

4.42.1 Detailed Description

Definition at line 9 of file SAccessibilityTranscriptionVis.h.

4.42.2 Constructor & Destructor Documentation

4.42.2.1 ∼SAccessibilityTranscriptionVis()

```
SAccessibilityTranscriptionVis::∼SAccessibilityTranscriptionVis ( )
```

Definition at line 5 of file SAccessibilityTranscriptionVis.cpp.

```
00006 {
00007 }
```

4.42.3 Member Function Documentation

4.42.3.1 Construct()

```
void SAccessibilityTranscriptionVis::Construct (
              const FArguments & InArgs )
```

Definition at line 9 of file SAccessibilityTranscriptionVis.cpp.

```
{
    // Transcription Holder
    TSharedPtr<SVerticalBox> TranscriptionHolder = SNew(SVerticalBox)
        + SVerticalBox::Slot()
        .Padding(4.0f)
        .AutoHeight();

    // Verify a least one slot will be constructed
    int TranscriptionSlotAmount = FMath::Max(1, InArgs._VisAmount);

    FSlateFontInfo FontInfo = FAppStyle::GetFontStyle("NormalText");
    FontInfo.Size = 12;

    TSharedPtr<STextBlock> CurrentTranscriptionSlot;
    for (int i = 0; i < TranscriptionSlotAmount; i++)
    {
        TranscriptionHolder->AddSlot()
            .HAlign(HAlign_Center)
            .Padding(4.0f)
            .AutoHeight()
            [
                SAssignNew(CurrentTranscriptionSlot, STextBlock)
                    .Text(FText::GetEmpty())
                    .Font(FontInfo)
                    .SimpleTextMode(true)
                    .ColorAndOpacity(i == 0 ? FSlateColor(FLinearColor(1.0f, 1.0f, 0, 1.0f)) :
      FSlateColor(FLinearColor(0.5f, 0.5f, 0.5f, 1.0f)))
            ];

        TranscriptionSlots.Add(CurrentTranscriptionSlot);
    }

    // Construct the Main Component

    ChildSlot
    .Padding(FMargin(5.0f))
    [
        SNew(SOverlay)
        + SOverlay::Slot()
        .ZOrder(1)
        [
            SNew(SBorder)
            .BorderBackgroundColor(FSlateColor(FLinearColor::Gray))
            [
                SNew(SBox)
                .MinDesiredWidth(250.0f)
                .MinDesiredHeight(60.0f)
                [
                    TranscriptionHolder.ToSharedRef()
                ]
            ]
        ]
    ];

    this->TranscriptionContainer = TranscriptionHolder;
}
```

4.42.3.2 SLATE_BEGIN_ARGS()

```
SAccessibilityTranscriptionVis::SLATE_BEGIN_ARGS (
              SAccessibilityTranscriptionVis ) [inline]
```

Definition at line 13 of file SAccessibilityTranscriptionVis.h.

```
    : _VisAmount(1)
    {}
```

4.42.3.3 Tick()

```
void SAccessibilityTranscriptionVis::Tick (
            const FGeometry & AllottedGeometry,
            const double InCurrentTime,
            const float InDeltaTime ) [override], [virtual]
```

Definition at line 66 of file SAccessibilityTranscriptionVis.cpp.

```
{
    SBox::Tick(AllottedGeometry, InCurrentTime, InDeltaTime);
}
```

4.42.3.4 UpdateTopTranscription()

```
void SAccessibilityTranscriptionVis::UpdateTopTranscription (
            const FString & InTopTranscription )
```

Updates the Top Transcription Text, shifting all current transcriptions down.

Definition at line 71 of file SAccessibilityTranscriptionVis.cpp.

```
{
    FString LastTopText = InTopTranscription;
    FString TempText;

    TSharedPtr<STextBlock> CurrentTranscriptionSlot;
    for (TWeakPtr<STextBlock>& TranscriptionSlot : TranscriptionSlots)
    {
        CurrentTranscriptionSlot = TranscriptionSlot.Pin();

        TempText = FString(CurrentTranscriptionSlot->GetText().ToString());
        CurrentTranscriptionSlot->SetText(FText::FromString(LastTopText));

        CurrentTranscriptionSlot->Invalidate(EInvalidateWidgetReason::PaintAndVolatility);

        LastTopText = TempText;
    }

    TranscriptionContainer.Pin()->Invalidate(EInvalidateWidget::Layout);
}
```

4.42.4 Member Data Documentation

4.42.4.1 TranscriptionContainer

```
TWeakPtr<SVerticalBox> SAccessibilityTranscriptionVis::TranscriptionContainer [protected]
```

The Container of the Transcription Slots.

Definition at line 39 of file SAccessibilityTranscriptionVis.h.

4.42.4.2 TranscriptionSlots

`TArray<TWeakPtr<STextBlock> > SAccessibilityTranscriptionVis::TranscriptionSlots [protected]`

Array of the created Transcription Slots, displaying recieved transcriptions.

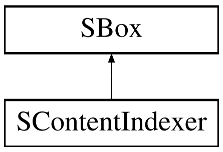
Definition at line 44 of file SAccessibilityTranscriptionVis.h.

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

- Source/OpenAccessibility/Public/AccessibilityWidgets/SAccessibilityTranscriptionVis.h
- Source/OpenAccessibility/Private/AccessibilityWidgets/SAccessibilityTranscriptionVis.cpp

4.43 SContentIndexer Class Reference

Inheritance diagram for SContentIndexer:



Public Member Functions

- SLATE_BEGIN_ARGS (SContentIndexer)
- **SLATE_PRIVATE_ATTRIBUTE_VARIABLE** (EVisibility, IndexVisibility)
- void Construct (const FArguments &InArgs)
- virtual void Tick (const FGeometry &AllottedGeometry, const double InCurrentTime, const float InDeltaTime) override
- void UpdateIndex (const int32 IndexValue)

  Updates the Index Value Displayed on the Indexer Text Widget.
- TSharedRef< SWidget > GetContent () const

  Gets the Current Content Being Indexed.
- template<typename CastType >
  TSharedRef< CastType > GetContent () const

  Gets the Current Content Being Indexed and Casts it to the Provided Type.

Protected Member Functions

- TSharedPtr< SWidget > ConstructTopIndexer (const FArguments &InArgs)

  Constructs the Indexer Widget with the Index on Top of the Content.
- TSharedPtr< SWidget > ConstructBottomIndexer (const FArguments &InArgs)

  Constructs the Indexer Widget with the Index Below the Content.
- TSharedPtr< SWidget > ConstructLeftIndexer (const FArguments &InArgs)

  Constructs the Indexer Widget with the Index to the Left of the Content.
- TSharedPtr< SWidget > ConstructRightIndexer (const FArguments &InArgs)

  Constructs the Indexer Widget with the Index to the Right of the Content.
- TSharedPtr< SWidget > ConstructContentContainer (TSharedRef< SWidget > ContentToIndex)

  Constructs the Container for the Indexer witht the provided Content.
- TSharedPtr< SWidget > ConstructIndexContainer (const FArguments &InArgs, FLinearColor Text↩
  Color=FLinearColor::White)

  Constructs the Indexer Widget with the provided Index Value.
- FText ConstructIndexText (int32 Index)

  Creates the Text Element of the Provided Index Value.

Protected Attributes

- TWeakPtr< SWidget > IndexedContent

  The Content That The Indexer Is Currently Indexing.
- TWeakPtr< class SIndexer > IndexerWidget

  The Text Block that Displays the Index Value.

4.43.1 Detailed Description

Definition at line 16 of file SContentIndexer.h.

4.43.2 Constructor & Destructor Documentation

4.43.2.1 ∼SContentIndexer()

```
SContentIndexer::∼SContentIndexer ( )
```

Definition at line 6 of file SContentIndexer.cpp.

```
{
}
```

4.43.3 Member Function Documentation

4.43.3.1 Construct()

```
void SContentIndexer::Construct (
              const FArguments & InArgs )
```

Definition at line 10 of file SContentIndexer.cpp.

```
{
    TSharedPtr<SWidget> Content;

    switch (InArgs._IndexPositionToContent)
    {
        case EIndexerPosition::Top:
            Content = ConstructTopIndexer(InArgs);
            break;

        case EIndexerPosition::Bottom:
            Content = ConstructBottomIndexer(InArgs);
            break;

        default:
        case EIndexerPosition::Left:
            Content = ConstructLeftIndexer(InArgs);
            break;

        case EIndexerPosition::Right:
            Content = ConstructRightIndexer(InArgs);
            break;
    }

    ChildSlot
    [
        Content.ToSharedRef()
    ];
}
```

4.43.3.2 ConstructBottomIndexer()

```
TSharedPtr< SWidget > SContentIndexer::ConstructBottomIndexer (
            const FArguments & InArgs )  [protected]
```

Constructs the Indexer Widget with the Index Below the Content.

Parameters

IndexValue	The Index Value to Index.
ContentToIndex	The Content that the Indexer is Wrapping.

Returns

Definition at line 74 of file SContentIndexer.cpp.

```
{
    return SNew(SVerticalBox)
    .Visibility(AccessWidgetVisibilityAttribute(InArgs._ContentToIndex.ToSharedRef()))

        + SVerticalBox::Slot()
        .HAlign(HAlign_Center)
        .VAlign(VAlign_Center)
        .AutoHeight()
        [
            ConstructContentContainer(InArgs._ContentToIndex.ToSharedRef()).ToSharedRef()
        ]

        + SVerticalBox::Slot()
        .HAlign(HAlign_Center)
        .VAlign(VAlign_Center)
        .AutoHeight()
        .Padding(.1f, .25f)
        [
            ConstructIndexContainer(InArgs).ToSharedRef()
        ];
}
```

4.43.3.3 ConstructContentContainer()

```
TSharedPtr< SWidget > SContentIndexer::ConstructContentContainer (
            TSharedRef< SWidget > ContentToIndex )  [protected]
```

Constructs the Container for the Indexer witht the provided Content.

Parameters

ContentToIndex	The Content that needs to be wrapped with an Indexer Widget.

Returns

Definition at line 143 of file SContentIndexer.cpp.

```
{
    IndexedContent = ContentToIndex;
    return ContentToIndex;
}
```

4.43.3.4 ConstructIndexContainer()

```
TSharedPtr< SWidget > SContentIndexer::ConstructIndexContainer (
              const FArguments & InArgs,
              FLinearColor TextColor = FLinearColor::White )  [protected]
```

Constructs the Indexer Widget with the provided Index Value.

Parameters

IndexValue	The Index Value to be displayed in the Indexer Widget.
TextColor	The Color of the Text displaying the Index.

Returns

Definition at line 149 of file SContentIndexer.cpp.

```
{
    return SAssignNew(IndexerWidget, SIndexer)
    .TextColor(TextColor)
    .BorderColor(FLinearColor::Gray)
    .IndexValue(InArgs._IndexValue)
    .IndexVisibility(InArgs._IndexVisibility);
}
```

4.43.3.5 ConstructIndexText()

```
FText SContentIndexer::ConstructIndexText (
              int32 Index )  [protected]
```

Creates the Text Element of the Provided Index Value.

Parameters

Index	The Index to convert into Text.

Returns

Definition at line 158 of file SContentIndexer.cpp.

```
{
    return FText::FromString(FString::FromInt(Index));
}
```

4.43.3.6 ConstructLeftIndexer()

```
TSharedPtr< SWidget > SContentIndexer::ConstructLeftIndexer (
            const FArguments & InArgs ) [protected]
```

Constructs the Indexer Widget with the Index to the Left of the Content.

Parameters

IndexValue	The Index Value to Index.
ContentToIndex	The Content that the Indexer is Wrapping.

Returns

Definition at line 97 of file SContentIndexer.cpp.

```
{
    return SNew(SHorizontalBox)
    .Visibility(AccessWidgetVisibilityAttribute(InArgs._ContentToIndex.ToSharedRef()))

        + SHorizontalBox::Slot()
        .VAlign(VAlign_Center)
        .HAlign(HAlign_Center)
        .AutoWidth()
        .Padding(.25f, .1f)
        [
            ConstructIndexContainer(InArgs).ToSharedRef()
        ]

        + SHorizontalBox::Slot()
        .VAlign(VAlign_Center)
        .HAlign(HAlign_Center)
        .AutoWidth()
        [
            ConstructContentContainer(InArgs._ContentToIndex.ToSharedRef()).ToSharedRef()
        ];
}
```

4.43.3.7 ConstructRightIndexer()

```
TSharedPtr< SWidget > SContentIndexer::ConstructRightIndexer (
            const FArguments & InArgs ) [protected]
```

Constructs the Indexer Widget with the Index to the Right of the Content.

Parameters

IndexValue	The Index Value to Index.
ContentToIndex	The Content that the Indexer is Wrapping.

Returns

Definition at line 120 of file SContentIndexer.cpp.

```
00121 {
00122     return SNew(SHorizontalBox)
00123     .Visibility(AccessWidgetVisibilityAttribute(InArgs._ContentToIndex.ToSharedRef()))
00124 
00125         + SHorizontalBox::Slot()
00126         .VAlign(VAlign_Center)
00127         .HAlign(HAlign_Center)
00128         .AutoWidth()
00129         [
00130             ConstructContentContainer(InArgs._ContentToIndex.ToSharedRef()).ToSharedRef()
00131         ]
00132 
00133         + SHorizontalBox::Slot()
00134         .VAlign(VAlign_Center)
00135         .HAlign(HAlign_Center)
00136         .AutoWidth()
00137         .Padding(.25f, .1f)
00138         [
00139             ConstructIndexContainer(InArgs).ToSharedRef()
00140         ];
00141 }
```

4.43.3.8 ConstructTopIndexer()

```
TSharedPtr< SWidget > SContentIndexer::ConstructTopIndexer (
              const FArguments & InArgs )  [protected]
```

Constructs the Indexer Widget with the Index on Top of the Content.

Parameters

IndexValue	The Index Value to Index.
ContentToIndex	The Content that the Indexer is Wrapping.

Returns

Definition at line 51 of file SContentIndexer.cpp.

```
00052 {
00053     return SNew(SVerticalBox)
00054     .Visibility(AccessWidgetVisibilityAttribute(InArgs._ContentToIndex.ToSharedRef()))
00055 
00056         + SVerticalBox::Slot()
00057         .HAlign(HAlign_Center)
00058         .VAlign(VAlign_Center)
00059         .AutoHeight()
00060         .Padding(.1f, .25f)
00061         [
00062             ConstructIndexContainer(InArgs).ToSharedRef()
00063         ]
00064 
00065         + SVerticalBox::Slot()
00066         .HAlign(HAlign_Center)
00067         .VAlign(VAlign_Center)
00068         .AutoHeight()
00069         [
00070             ConstructContentContainer(InArgs._ContentToIndex.ToSharedRef()).ToSharedRef()
00071         ];
00072 }
```

4.43.3.9 GetContent() [1/2]

```
TSharedRef< SWidget > SContentIndexer::GetContent ( ) const  [inline]
```

Gets the Current Content Being Indexed.

Returns

A Shared Ptr of the Indexed Content

Definition at line 54 of file SContentIndexer.h.

```
    {
        return IndexedContent.Pin().ToSharedRef();
    }
```

4.43.3.10 GetContent() [2/2]

```
template<typename CastType >
TSharedRef< CastType > SContentIndexer::GetContent ( ) const  [inline]
```

Gets the Current Content Being Indexed and Casts it to the Provided Type.

Template Parameters

CastType	The Type To Cast The Stored Value To.

Returns

The Casted SharedReference.

Definition at line 66 of file SContentIndexer.h.

```
    {
        return CastStaticSharedPtr<CastType>(IndexedContent.Pin());
    }
```

4.43.3.11 SLATE_BEGIN_ARGS()

```
SContentIndexer::SLATE_BEGIN_ARGS (
            SContentIndexer  )  [inline]
```

Definition at line 20 of file SContentIndexer.h.

```
        : _IndexValue(0)
        , _IndexPositionToContent(EIndexerPosition::Left)
        , _ContentToIndex(SNullWidget::NullWidget)
        {}
```

4.43.3.12 Tick()

```
void SContentIndexer::Tick (
            const FGeometry & AllottedGeometry,
            const double InCurrentTime,
            const float InDeltaTime )  [override], [virtual]
```

Definition at line 40 of file SContentIndexer.cpp.

```
{
    SBox::Tick(AllottedGeometry, InCurrentTime, InDeltaTime);
}
```

4.43.3.13 UpdateIndex()

```
void SContentIndexer::UpdateIndex (
            const int32 IndexValue )
```

Updates the Index Value Displayed on the Indexer Text Widget.

Parameters

IndexValue	The New Interger Index to Show.

Definition at line 45 of file SContentIndexer.cpp.

```
{
    if (IndexerWidget.IsValid())
        IndexerWidget.Pin()->UpdateIndex( IndexValue );
}
```

4.43.4 Member Data Documentation

4.43.4.1 IndexedContent

```
TWeakPtr<SWidget> SContentIndexer::IndexedContent  [protected]
```

The Content That The Indexer Is Currently Indexing.

Definition at line 132 of file SContentIndexer.h.

4.43.4.2 IndexerWidget

```
TWeakPtr<class SIndexer> SContentIndexer::IndexerWidget  [protected]
```

The Text Block that Displays the Index Value.

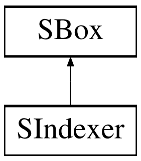
Definition at line 137 of file SContentIndexer.h.

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

- Source/OpenAccessibility/Public/AccessibilityWidgets/SContentIndexer.h
- Source/OpenAccessibility/Private/AccessibilityWidgets/SContentIndexer.cpp

4.44 SIndexer Class Reference

Inheritance diagram for SIndexer:



Public Member Functions

- SLATE_BEGIN_ARGS (SIndexer)
- **SLATE_PRIVATE_ARGUMENT_VARIABLE** (int32, IndexValue)
- **SLATE_PRIVATE_ATTRIBUTE_VARIABLE** (EVisibility, IndexVisibility)
- virtual void Tick (const FGeometry &AllotedGeometry, const double InCurrentTime, const float InDeltaTime) override
- void Construct (const FArguments &InArgs)
- void UpdateIndex (const int32 NewIndex)

  Updates the Index Widget with the New Index Value.
- void UpdateIndex (const FString &NewIndex)

  Updates the Index Widget with the New String Index Value.
- void UpdateIndex (const FText &NewIndex)

  Updates the Index Widget with the New Text Index Value.
- TSharedPtr< STextBlock > GetIndexText () const

  Gets the Index TextBlock Widget.

Protected Attributes

- TWeakPtr< STextBlock > IndexTextBlock

  Weak Pointer to the Main TextBlock Widget.

4.44.1 Detailed Description

Definition at line 7 of file SIndexer.h.

4.44.2 Constructor & Destructor Documentation

4.44.2.1 ∼SIndexer()

```
SIndexer::∼SIndexer ( )
```

Definition at line 5 of file SIndexer.cpp.

```
00006 {
00007
00008 }
```

4.44.3 Member Function Documentation

4.44.3.1 Construct()

```
void SIndexer::Construct (
            const FArguments & InArgs )
```

Definition at line 15 of file SIndexer.cpp.

```
{
    ChildSlot
    [
        SNew(SBorder)
        .HAlign(HAlign_Center)
        .VAlign(VAlign_Center)
        .Visibility(InArgs._IndexVisibility)
        .Padding(FMargin(4.f, 2.f))
        .BorderBackgroundColor( FSlateColor(InArgs._BorderColor) )
        [
            SAssignNew(IndexTextBlock, STextBlock)
            .Text( FText::FromString(FString::FromInt(InArgs._IndexValue)) )
            .TextShapingMethod( ETextShapingMethod::KerningOnly )
            .ColorAndOpacity( FSlateColor(InArgs._TextColor) )
        ]
    ];
}
```

4.44.3.2 GetIndexText()

```
TSharedPtr< STextBlock > SIndexer::GetIndexText ( ) const  [inline]
```

Gets the Index TextBlock Widget.

Returns

A Valid TextBlock Widget, if it is still found. Otherwise InValid SharedPtr.

Definition at line 55 of file SIndexer.h.

```
    {
        return IndexTextBlock.IsValid() ? IndexTextBlock.Pin() : TSharedPtr<STextBlock>();
    }
```

4.44.3.3 SLATE_BEGIN_ARGS()

```
SIndexer::SLATE_BEGIN_ARGS (
            SIndexer  )  [inline]
```

Definition at line 10 of file SIndexer.h.

```
    : _TextColor(FLinearColor::White)
    , _BorderColor(FLinearColor::Black)
    {}
```

4.44.3.4 Tick()

```
void SIndexer::Tick (
            const FGeometry & AllotedGeometry,
            const double InCurrentTime,
            const float InDeltaTime )  [override], [virtual]
```

Definition at line 10 of file SIndexer.cpp.

```
{
    SBox::Tick(AllotedGeometry, InCurrentTime, InDeltaTime);
}
```

4.44.3.5 UpdateIndex() [1/3]

```
void SIndexer::UpdateIndex (
            const FString & NewIndex )
```

Updates the Index Widget with the New String Index Value.

Parameters

NewIndex	- The New Index Value, in String Form.

Definition at line 44 of file SIndexer.cpp.

```
{
    if (!IndexTextBlock.IsValid())
        return;

    IndexTextBlock.Pin()->SetText(
        FText::FromString(NewIndex)
    );
}
```

4.44.3.6 UpdateIndex() [2/3]

```
void SIndexer::UpdateIndex (
            const FText & NewIndex )
```

Updates the Index Widget with the New Text Index Value.

Parameters

NewIndex	- The New Index Value, in Text Form.

Definition at line 54 of file SIndexer.cpp.

```
{
    if (!IndexTextBlock.IsValid())
        return;

    IndexTextBlock.Pin()->SetText(NewIndex);
}
```

4.44.3.7 UpdateIndex() [3/3]

```
void SIndexer::UpdateIndex (
            const int32 NewIndex )
```

Updates the Index Widget with the New Index Value.

Parameters

NewIndex	- The New Index Value.

Definition at line 34 of file SIndexer.cpp.

```
{
    if (!IndexTextBlock.IsValid())
        return;

    IndexTextBlock.Pin()->SetText(
        FText::FromString( FString::FromInt(NewIndex) )
    );
}
```

4.44.4 Member Data Documentation

4.44.4.1 IndexTextBlock

```
TWeakPtr<STextBlock> SIndexer::IndexTextBlock  [protected]
```

Weak Pointer to the Main TextBlock Widget.

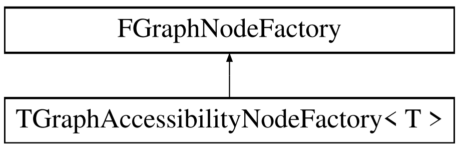
Definition at line 65 of file SIndexer.h.

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

- Source/OpenAccessibility/Public/AccessibilityWidgets/SIndexer.h
- Source/OpenAccessibility/Private/AccessibilityWidgets/SIndexer.cpp

4.45 TGraphAccessibilityNodeFactory< T > Class Template Reference

Inheritance diagram for TGraphAccessibilityNodeFactory< T >:



Public Member Functions

- TGraphAccessibilityNodeFactory (TSharedRef< FAssetAccessibilityRegistry > InAccessibilityRegistry)
- virtual TSharedPtr< class SGraphNode > CreateNodeWidget (UEdGraphNode ∗InNode) override
- virtual TSharedPtr< class SGraphPin > CreatePinWidget (UEdGraphPin ∗InPin) override

Protected Attributes

- TSharedRef< FAssetAccessibilityRegistry > AccessibilityRegistry
- TSharedPtr< T > Implementation

4.45.1 Detailed Description

template<class T>
class TGraphAccessibilityNodeFactory< T >

Definition at line 17 of file AccessibilityNodeFactory.h.

4.45.2 Constructor & Destructor Documentation

4.45.2.1 TGraphAccessibilityNodeFactory() [1/2]

```
template<class T >
TGraphAccessibilityNodeFactory< T >::TGraphAccessibilityNodeFactory ( )  [inline]
```

Definition at line 23 of file AccessibilityNodeFactory.h.

```
    {
        Implementation = TSharedPtr<T>();

        AccessibilityRegistry =
FOpenAccessibilityModule::Get().AssetAccessibilityRegistry.ToSharedRef();
    }
```

4.45.2.2 TGraphAccessibilityNodeFactory() [2/2]

```
template<class T >
TGraphAccessibilityNodeFactory< T >::TGraphAccessibilityNodeFactory (
            TSharedRef< FAssetAccessibilityRegistry > InAccessibilityRegistry )  [inline]
```

Definition at line 30 of file AccessibilityNodeFactory.h.

```
        : AccessibilityRegistry(InAccessibilityRegistry)
    {
        Implementation = TSharedPtr<T>();
    }
```

4.45.2.3 ∼TGraphAccessibilityNodeFactory()

```
template<class T >
virtual TGraphAccessibilityNodeFactory< T >::∼TGraphAccessibilityNodeFactory ( ) [inline],
[virtual]
```

Definition at line 36 of file AccessibilityNodeFactory.h.

```
    {

    }
```

4.45.3 Member Function Documentation

4.45.3.1 CreateNodeWidget()

```
template<class T >
TSharedPtr< class SGraphNode > TGraphAccessibilityNodeFactory< T >::CreateNodeWidget (
              UEdGraphNode * InNode ) [override], [virtual]
```

Creates a Visual Node Widget from the Provided Node Object.

Parameters

InNode	The Node To Create a Node Widget From.

Returns

Definition at line 70 of file AccessibilityNodeFactory.h.

```
{
    check(InNode != nullptr);

    TSharedPtr<SGraphNode> OutNode = Implementation->CreateNodeWidget(InNode);

    // Apply Accessibility Visuals to the Node.

    TSharedRef<FGraphIndexer> GraphIndexer =
 AccessibilityRegistry->GetGraphIndexer(InNode->GetGraph());

    int NodeIndex = -1;
    GraphIndexer->GetOrAddNode(InNode);

    TSharedRef<SWidget> WidgetToWrap = OutNode->GetSlot(ENodeZone::Center)->GetWidget();

    check(WidgetToWrap != SNullWidget::NullWidget);

    OutNode->GetOrAddSlot(ENodeZone::Center)
        .HAlign(HAlign_Fill)
        [
            SNew(SVerticalBox)

                + SVerticalBox::Slot()
                .HAlign(HAlign_Fill)
                .AutoHeight()
                .Padding(FMargin(1.5f, 0.25f))
                [
                    SNew(SOverlay)

                        + SOverlay::Slot()
                        [
                            SNew(SImage)
                                .Image(FAppStyle::Get().GetBrush("Graph.Node.Body"))
                        ]

                        + SOverlay::Slot()
                        .Padding(FMargin(4.0f, 0.0f))
                        [
                            SNew(SHorizontalBox)
                                + SHorizontalBox::Slot()
                                .HAlign(HAlign_Right)
                                .VAlign(VAlign_Center)
                                .Padding(1.f)
                                [
                                    SNew(SOverlay)
                                        + SOverlay::Slot()
                                        [
                                            SNew(SIndexer)
                                                .IndexValue(NodeIndex)
                                                .TextColor(FLinearColor::White)
                                                .BorderColor(FLinearColor::Gray)
                                        ]
                                ]
                        ]
                ]

                + SVerticalBox::Slot()
                .HAlign(HAlign_Fill)
                .AutoHeight()
                [
                    WidgetToWrap
                ]
        ];

    return OutNode;
}
```

4.45.3.2 CreatePinWidget()

template<class T >
TSharedPtr< class SGraphPin > TGraphAccessibilityNodeFactory< T >::CreatePinWidget (
UEdGraphPin * *InPin*) [override], [virtual]

Creates a Visual Pin Widget from the Provided Pin Object.

Parameters

InPin	The Pin to Create a Pin Widget From.

Returns

Definition at line 138 of file AccessibilityNodeFactory.h.

```
{
    check(InPin != nullptr);

    TSharedPtr<SGraphPin> OutPin = Implementation->CreatePinWidget(InPin);
    SGraphPin* OutPinPtr = OutPin.Get();

    TSharedRef<FGraphIndexer> GraphIndexer =
  AccessibilityRegistry->GetGraphIndexer(InPin->GetOwningNode()->GetGraph());

    int PinIndex = -1;
    PinIndex = InPin->GetOwningNode()->GetPinIndex(InPin);

    TSharedRef<SWidget> AccessiblityWidget = SNew(SOverlay)
        .Visibility_Lambda([OutPinPtr]() -> EVisibility {
            if (OutPinPtr->HasAnyUserFocusOrFocusedDescendants() || OutPinPtr->IsHovered())
                return EVisibility::Visible;

            return EVisibility::Hidden;
        })
        + SOverlay::Slot()
        [
            SNew(STextBlock)
                .ColorAndOpacity(FLinearColor::White)
                .ShadowColorAndOpacity(FLinearColor::Black)
                .ShadowOffset(FIntPoint(-1, 1))
                .Font(FAppStyle::Get().GetFontStyle("Graph.Node.Pin.Font"))
                .Text(FText::FromString("[" + FString::FromInt(PinIndex) + "]"))
        ];

    // Get Pin Widget Content, before modifying it.
    TSharedRef<SWidget> PinWidgetContent = OutPin->GetContent();

    // Modify the Pin Widget Content, based on the Pin's Direction.
    switch (OutPin->GetDirection())
    {
    case EEdGraphPinDirection::EGPD_Input:
    {
        OutPin->SetContent(
            SNew(SHorizontalBox)
            + SHorizontalBox::Slot()
            [
                PinWidgetContent
            ]
            + SHorizontalBox::Slot()
            [
                AccessiblityWidget
            ]
        );

        break;
    }

    case EEdGraphPinDirection::EGPD_Output:
    {
        OutPin->SetContent(
            SNew(SHorizontalBox)
            + SHorizontalBox::Slot()
            .AutoWidth()
            [
                AccessiblityWidget
            ]
            + SHorizontalBox::Slot()
            .AutoWidth()
            [
                PinWidgetContent
            ]
        );

        break;
    }
    }

    return OutPin;
}
```

4.45.4 Member Data Documentation

4.45.4.1 AccessibilityRegistry

`template<class T >`
`TSharedRef<FAssetAccessibilityRegistry> TGraphAccessibilityNodeFactory< T >::Accessibility↩ Registry [protected]`

The Asset Registry of the Open Accessibility Plugin.

Definition at line 64 of file AccessibilityNodeFactory.h.

4.45.4.2 Implementation

```
template<class T >
TSharedPtr<T> TGraphAccessibilityNodeFactory< T >::Implementation [protected]
```

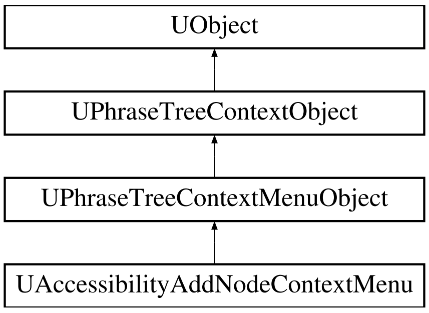
Definition at line 66 of file AccessibilityNodeFactory.h.

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

- Source/OpenAccessibility/Public/AccessibilityNodeFactory.h

4.46 UAccessibilityAddNodeContextMenu Class Reference

Inheritance diagram for UAccessibilityAddNodeContextMenu:


UObject
UPhraseTreeContextObject
UPhraseTreeContextMenuObject
UAccessibilityAddNodeContextMenu

Public Member Functions

- UAccessibilityAddNodeContextMenu (TSharedRef< IMenu > Menu)
- UAccessibilityAddNodeContextMenu (TSharedRef< IMenu > Menu, TSharedRef< SGraphActionMenu > GraphMenu)
- UAccessibilityAddNodeContextMenu (TSharedRef< IMenu > Menu, TSharedRef< SGraphActionMenu > GraphMenu, TSharedRef< STreeView< TSharedPtr< FGraphActionNode > > > TreeView)
- virtual void Init (TSharedRef< IMenu > InMenu, TSharedRef< FPhraseNode > InContextRoot) override

    Initializes the Context Menu.
- void Init (TSharedRef< IMenu > InMenu, TSharedRef< SGraphActionMenu > InGraphMenu, TSharedRef< STreeView< TSharedPtr< FGraphActionNode > > > InTreeView)

    Initializes the Context Menu from the given components.
- virtual void Init (TSharedRef< IMenu > InMenu) override

    Initializes the Context Menu from the given components.
- virtual bool Tick (float DeltaTime) override
- virtual bool Close () override

    Closes the Context Menu.
- virtual void ScaleMenu (const float ScaleFactor=1.5f) override

    Scaled the Context Menu's Core Components based on the provided ScaleFactor.
- bool DoesItemsRequireRefresh ()

    Does the Context Menu's TreeView Require a Refresh of Accessibility Widgets.
- void RefreshAccessibilityWidgets ()

    Performs a Refresh of the TreeView's Accessibility Widgets.
- void GetGraphActionFromIndex (const int32 InIndex, FGraphActionNode *OutGraphAction)

Gets the GraphActionNode from the given Index.

- FGraphActionNode ∗ GetGraphActionFromIndex (const int32 InIndex)

    Gets the GraphActionNode from the given Index.
- TSharedPtr< FGraphActionNode > GetGraphActionFromIndexSP (const int32 InIndex)

    Gets the GraphActionNode from the given Index.
- void SelectGraphAction (const int32 InIndex)

    Performs a Selction in the TreeView, based on the given Index.
- void PerformGraphAction (const int32 InIndex)

    Performs the Action to the Linked GraphActionNode, based on the given Index.
- FString GetFilterText ()

    Gets the Current Filter Text in the Search Bar.
- void SetFilterText (const FString &InFilterText)

    Overrides the Current Filter Text with the given string.
- void AppendFilterText (const FString &InFilterText)

    Append the given string to the End of the Current Filter Text.
- void ResetFilterText ()

    Clears the Current Filter Text.
- void SetScrollDistance (const float InScrollDistance)

    Sets the Scroll Distance of the TreeView.
- void AppendScrollDistance (const float InScrollDistance)

    Adds the provided value to the Current Scroll Distance.
- void SetScrollDistanceTop ()

    Sets the Scroll Distance to the Top of the TreeView. Taking the View to the First Item in the TreeView.
- void SetScrollDistanceBottom ()

    Sets the Scroll Distance to the Bottom of the TreeView. Taking the View to the Last Item in the TreeView.
- void ToggleContextAwareness ()

    Toggles the Context Awareness of the Node List.

Public Attributes

- TWeakPtr< SGraphActionMenu > GraphMenu

    The SGraphActionMenu for the Context Menu.
- TWeakPtr< STreeView< TSharedPtr< FGraphActionNode > > > TreeView

    The STreeView for the Context Menu.
- TWeakPtr< SEditableTextBox > FilterTextBox

    The SEditableTextBox for the Context Menu. Used for Filtering through GraphNodes.
- TWeakPtr< SCheckBox > ContextAwarenessCheckBox

    The Context Awareness CheckBox for the Context Menu. Used for toggling Context Awareness, in searching for GraphNodes.

Protected Member Functions

- void ApplyAccessibilityWidget (TSharedRef< STableRow< TSharedPtr< FGraphActionNode > > > Item←Widget)

    Applies the Accessibility Widget to the given Item's TableRow Widget.
- void UpdateAccessibilityWidget (TSharedRef< STableRow< TSharedPtr< FGraphActionNode > > > ItemWidget)

    Updates the previously applied Accessibility Widget, with the new index.

Protected Attributes

- FString PrevFilterString
- int32 PrevNumItemsBeingObserved
- int32 PrevNumGeneratedChildren
- double PrevScrollDistance

4.46.1 Detailed Description

Definition at line 17 of file AccessibilityAddNodeContextMenu.h.

4.46.2 Constructor & Destructor Documentation

4.46.2.1 UAccessibilityAddNodeContextMenu() [1/4]

```
UAccessibilityAddNodeContextMenu::UAccessibilityAddNodeContextMenu ( )
```

Definition at line 13 of file AccessibilityAddNodeContextMenu.cpp.

```
    : UPhraseTreeContextMenuObject()
{

}
```

4.46.2.2 UAccessibilityAddNodeContextMenu() [2/4]

```
UAccessibilityAddNodeContextMenu::UAccessibilityAddNodeContextMenu (
            TSharedRef< IMenu > Menu )
```

Definition at line 19 of file AccessibilityAddNodeContextMenu.cpp.

```
    : UPhraseTreeContextMenuObject(Menu)
{

}
```

4.46.2.3 UAccessibilityAddNodeContextMenu() [3/4]

```
UAccessibilityAddNodeContextMenu::UAccessibilityAddNodeContextMenu (
            TSharedRef< IMenu > Menu,
            TSharedRef< SGraphActionMenu > GraphMenu )
```

Definition at line 25 of file AccessibilityAddNodeContextMenu.cpp.

```
: UPhraseTreeContextMenuObject(Menu)
{
    this->GraphMenu = GraphMenu;
    this->FilterTextBox = GraphMenu->GetFilterTextBox();
}
```

4.46.2.4 UAccessibilityAddNodeContextMenu() [4/4]

```
UAccessibilityAddNodeContextMenu::UAccessibilityAddNodeContextMenu (
            TSharedRef< IMenu > Menu,
            TSharedRef< SGraphActionMenu > GraphMenu,
            TSharedRef< STreeView< TSharedPtr< FGraphActionNode > > > TreeView )
```

Definition at line 32 of file AccessibilityAddNodeContextMenu.cpp.

```
: UPhraseTreeContextMenuObject(Menu)
{
    this->GraphMenu = GraphMenu;
    this->TreeView = TreeView;
    this->FilterTextBox = GraphMenu->GetFilterTextBox();
}
```

4.46.2.5 ∼UAccessibilityAddNodeContextMenu()

```
UAccessibilityAddNodeContextMenu::∼UAccessibilityAddNodeContextMenu ( )
```

Definition at line 40 of file AccessibilityAddNodeContextMenu.cpp.

```
{

}
```

4.46.3 Member Function Documentation

4.46.3.1 AppendFilterText()

```
void UAccessibilityAddNodeContextMenu::AppendFilterText (
            const FString & InFilterText )
```

Append the given string to the End of the Current Filter Text.

Parameters

InFilterText	The Text To Append to the End.

Definition at line 282 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    FilterTextBox.Pin()->SetText(
        FText::FromString(
            FilterTextBox.Pin()->GetText().ToString() + TEXT(" ") + InFilterText
        )
    );
}
```

4.46.3.2 AppendScrollDistance()

```
void UAccessibilityAddNodeContextMenu::AppendScrollDistance (
            const float InScrollDistance )
```

Adds the provided value to the Current Scroll Distance.

Parameters

InScrollDistance	The Scroll Distance to Add the Current Distance. Positive Values are down, with Negative being up.

Definition at line 301 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    if (TreeView.Pin()->GetScrollOffset() + InScrollDistance < 0.0f)
    {
        TreeView.Pin()->SetScrollOffset(0.0f);
        return;
    }

    TreeView.Pin()->AddScrollOffset(InScrollDistance, true);
}
```

4.46.3.3 ApplyAccessibilityWidget()

```
void UAccessibilityAddNodeContextMenu::ApplyAccessibilityWidget (
            TSharedRef< STableRow< TSharedPtr< FGraphActionNode > > > ItemWidget )  [protected]
```

Applies the Accessibility Widget to the given Item's TableRow Widget.

Parameters

Item	The Item to apply to.
ItemWidget	The Items linked widget.

Definition at line 327 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    TSharedPtr<SWidget> ItemContent = ItemWidget->GetContent();

    ItemWidget->SetContent(
        SNew(SContentIndexer)
        .IndexValue(ItemWidget->GetIndexInList())
        .IndexPositionToContent(EIndexerPosition::Left)
        .ContentToIndex(ItemContent)
    );
}
```

4.46.3.4 Close()

```
bool UAccessibilityAddNodeContextMenu::Close ( )  [override], [virtual]
```

Closes the Context Menu.

Returns

Returns True if the Menu was Successfully closed.

Reimplemented from UPhraseTreeContextMenuObject.

Definition at line 131 of file AccessibilityAddNodeContextMenu.cpp.
```
{
    RemoveTickDelegate();
    Menu.Pin()->Dismiss();

    return true;
}
```

4.46.3.5 DoesItemsRequireRefresh()

`bool UAccessibilityAddNodeContextMenu::DoesItemsRequireRefresh ( )`

Does the Context Menu's TreeView Require a Refresh of Accessibility Widgets.

Returns

Returns True if the Context Menu requires change.

Definition at line 157 of file AccessibilityAddNodeContextMenu.cpp.
```
{
    TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

    return (
        FilterTextBox.Pin()->GetText().ToString() != PrevFilterString ||
        TreeViewPtr->GetNumItemsBeingObserved() != PrevNumItemsBeingObserved ||
        TreeViewPtr->GetNumGeneratedChildren() != PrevNumGeneratedChildren ||
        TreeViewPtr->GetScrollDistance().Y != PrevScrollDistance
    );
}
```

4.46.3.6 GetFilterText()

`FString UAccessibilityAddNodeContextMenu::GetFilterText ( )`

Gets the Current Filter Text in the Search Bar.

Returns

The Current Filter Text in the Search Bar.

Definition at line 272 of file AccessibilityAddNodeContextMenu.cpp.
```
{
    return FilterTextBox.Pin()->GetText().ToString();
}
```

4.46.3.7 GetGraphActionFromIndex() [1/2]

```
FGraphActionNode * UAccessibilityAddNodeContextMenu::GetGraphActionFromIndex (
            const int32 InIndex )
```

Gets the GraphActionNode from the given Index.

Parameters

InIndex	The Index of the Node to Find.

Returns

The Found GraphActionNode for the Index, or nullptr.

Definition at line 207 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    TArrayView<const TSharedPtr<FGraphActionNode» Items = TreeView.Pin()->GetItems();

    if (Items.Num() > InIndex)
        return Items[InIndex].Get();

    else return nullptr;
}
```

4.46.3.8 GetGraphActionFromIndex() [2/2]

```
void UAccessibilityAddNodeContextMenu::GetGraphActionFromIndex (
            const int32 InIndex,
            FGraphActionNode * OutGraphAction )
```

Gets the GraphActionNode from the given Index.

Parameters

InIndex	The Index of the Node to Find.
OutGraphAction	The Found GraphActionNode for the Index, or nullptr.

Definition at line 217 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    TArrayView<const TSharedPtr<FGraphActionNode» Items = TreeView.Pin()->GetItems();

    if (Items.Num() > InIndex)
        OutGraphAction = Items[InIndex].Get();

    else OutGraphAction = nullptr;
}
```

4.46.3.9 GetGraphActionFromIndexSP()

```
TSharedPtr< FGraphActionNode > UAccessibilityAddNodeContextMenu::GetGraphActionFromIndexSP (
            const int32 InIndex )
```

Gets the GraphActionNode from the given Index.

Parameters

InIndex	The Index of the Node to Find.

Returns

The Found GraphActionNode for the Index, or nullptr.

Definition at line 227 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    if (TreeView.Pin()->GetItems().Num() <= InIndex)
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("GetGraphActionFromIndexSP: Provided Index is Out
    of Range."));
        return nullptr;
    }
    return TreeView.Pin()->GetItems()[InIndex];
}
```

4.46.3.10 Init() [1/3]

```
void UAccessibilityAddNodeContextMenu::Init (
            TSharedRef< IMenu > InMenu )  [override], [virtual]
```

Initializes the Context Menu from the given components.

Parameters

InMenu	

Reimplemented from UPhraseTreeContextMenuObject.

Definition at line 52 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    UPhraseTreeContextMenuObject::Init(InMenu);

    // This is a Mess but half the Menu Containers are private, so have to move myself to key aspects
    of the Menu.

    TSharedPtr<SWidget> KeyboardFocusedWidget = StaticCastSharedPtr<SEditableText>(
        FSlateApplication::Get().GetKeyboardFocusedWidget()
    );
    if (!KeyboardFocusedWidget.IsValid())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("AddNodeContextWrapper::Init: KeyboardFocusedWidget
    is Invalid."));
        return;
    }

    this->GraphMenu = StaticCastSharedPtr<SGraphActionMenu>(
        KeyboardFocusedWidget
        ->GetParentWidget()
        ->GetParentWidget()
        ->GetParentWidget()
        ->GetParentWidget()
        ->GetParentWidget()
    );

    {
        TSharedPtr<SSearchBox> SearchBox = StaticCastSharedPtr<SSearchBox>(
            KeyboardFocusedWidget
                ->GetParentWidget()
                ->GetParentWidget()
                ->GetParentWidget()
        );

        TSharedRef<SWidget> SearchBoxSibling =
    SearchBox->GetParentWidget()->GetChildren()->GetChildAt(1);
        this->TreeView = StaticCastSharedRef<STreeView<TSharedPtr<FGraphActionNode>>>(
            SearchBoxSibling->GetChildren()->GetChildAt(0)->GetChildren()->GetChildAt(0)
        );
    }

    {
        TSharedRef<SCheckBox> CheckBox = StaticCastSharedRef<SCheckBox>(

    this->GraphMenu.Pin()->GetParentWidget()->GetChildren()->GetChildAt(0)->GetChildren()->GetChildAt(2)
        );

        this->ContextAwarenessCheckBox = CheckBox;
    }

    this->FilterTextBox = this->GraphMenu.Pin()->GetFilterTextBox();

    FSlateApplication::Get().SetKeyboardFocus(this->TreeView.Pin());
}
```

4.46.3.11 Init() [2/3]

```
void UAccessibilityAddNodeContextMenu::Init (
            TSharedRef< IMenu > InMenu,
            TSharedRef< FPhraseNode > InContextRoot )  [override], [virtual]
```

Initializes the Context Menu.

Parameters

InMenu	The Menu to Initialize from and obtain key components.
InContextRoot	The Context Root in the PhraseTree that this ContextMenu Originates from.

Reimplemented from UPhraseTreeContextMenuObject.

Definition at line 45 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    Init(InMenu);

    this->ContextRoot = InContextRoot;
}
```

4.46.3.12 Init() [3/3]

```
void UAccessibilityAddNodeContextMenu::Init (
            TSharedRef< IMenu > InMenu,
            TSharedRef< SGraphActionMenu > InGraphMenu,
            TSharedRef< STreeView< TSharedPtr< FGraphActionNode > > > InTreeView )
```

Initializes the Context Menu from the given components.

Parameters

InMenu	The Menu Item, for the tragetContext Menu.
InGraphMenu	The GraphActionMenu, for the target Context Menu.
InTreeView	The GraphAction TreeView, for the target Context Menu.

Definition at line 103 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    UPhraseTreeContextMenuObject::Init(InMenu);
00106
00107     this->GraphMenu = InGraphMenu;
00108     this->TreeView = InTreeView;
00109     this->FilterTextBox = InGraphMenu->GetFilterTextBox();
00110 }
```

4.46.3.13 PerformGraphAction()

```
void UAccessibilityAddNodeContextMenu::PerformGraphAction (
              const int32 InIndex )
```

Performs the Action to the Linked GraphActionNode, based on the given Index.

Parameters

InIndex	

Definition at line 251 of file AccessibilityAddNodeContextMenu.cpp.

```
00252 {
00253     TSharedPtr<FGraphActionNode> GraphAction = GetGraphActionFromIndexSP(InIndex);
00254
00255     if (!GraphAction.IsValid())
00256     {
00257         UE_LOG(LogOpenAccessibility, Warning, TEXT("PerformGraphAction: Provided GraphAction is
       Invalid."));
00258     }
00259
00260     if (GraphAction->IsActionNode())
00261     {
00262         TreeView.Pin()->Private_ClearSelection();
00263         TreeView.Pin()->Private_SetItemSelection(GraphAction, true, true);
00264         TreeView.Pin()->Private_SignalSelectionChanged(ESelectInfo::OnMouseClick);
00265     }
00266     else
00267     {
00268         TreeView.Pin()->Private_OnItemDoubleClicked(GraphAction);
00269     }
00270 }
```

4.46.3.14 RefreshAccessibilityWidgets()

```
void UAccessibilityAddNodeContextMenu::RefreshAccessibilityWidgets ( )
```

Performs a Refresh of the TreeView's Accessibility Widgets.

Definition at line 169 of file AccessibilityAddNodeContextMenu.cpp.

```
00170 {
00171
00172     TSharedPtr<STreeView<TSharedPtr<FGraphActionNode>>> TreeViewPtr = TreeView.Pin();
00173
00174     TArray<TSharedPtr<FGraphActionNode>> Items =
       TArray<TSharedPtr<FGraphActionNode>>(TreeViewPtr->GetRootItems());
00175
00176     {
00177         TSharedPtr<STableRow<TSharedPtr<FGraphActionNode>>> ItemWidget = nullptr;
00178
00179         while (Items.Num() > 0)
00180         {
00181             const TSharedPtr<FGraphActionNode> Item = Items[0];
00182             Items.RemoveAt(0);
00183
00184             if (TreeViewPtr->IsItemExpanded(Item))
00185                 Items.Append(Item->Children);
```

```
            ItemWidget = StaticCastSharedPtr<STableRow<TSharedPtr<FGraphActionNode>»>(
                TreeViewPtr->WidgetFromItem(Item)
            );

            if (!ItemWidget.IsValid())
                continue;

            // TO-DO: Change To Non-HardCoded Type Comparison.
            if (ItemWidget->GetContent()->GetType() == "SContentIndexer")
            {
                UpdateAccessibilityWidget(ItemWidget.ToSharedRef());
            }
            else
            {
                ApplyAccessibilityWidget(ItemWidget.ToSharedRef());
            }
        }
    }
}
```

4.46.3.15 ResetFilterText()

`void UAccessibilityAddNodeContextMenu::ResetFilterText ( )`

Clears the Current Filter Text.

Definition at line 291 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    FilterTextBox.Pin()->SetText(FText::FromString(TEXT("")));
}
```

4.46.3.16 ScaleMenu()

```
void UAccessibilityAddNodeContextMenu::ScaleMenu (
               const float ScaleFactor = 1.5f )  [override], [virtual]
```

Scaled the Context Menu's Core Components based on the provided ScaleFactor.

Parameters

ScaleFactor	The Factor for Scaling the Context Menu.

Reimplemented from UPhraseTreeContextMenuObject.

Definition at line 139 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    // Scale TreeView Element
    {
        TSharedPtr<STreeView<TSharedPtr<FGraphActionNode>»> TreeViewPtr = TreeView.Pin();

        TreeViewPtr->SetItemHeight(16 * ScaleFactor);
    }

    // Scale Window Element
    {
        TSharedPtr<SWindow> WindowPtr = Window.Pin();

        WindowPtr->SetSizingRule(ESizingRule::UserSized);
        WindowPtr->Resize(WindowPtr->GetSizeInScreen() * ScaleFactor);
    }
00155 }
```

4.46.3.17 SelectGraphAction()

```
void UAccessibilityAddNodeContextMenu::SelectGraphAction (
            const int32 InIndex )
```

Performs a Selction in the TreeView, based on the given Index.

Parameters

InIndex	

Definition at line 237 of file AccessibilityAddNodeContextMenu.cpp.

```
00238 {
00239     TSharedPtr<FGraphActionNode> GraphAction = GetGraphActionFromIndexSP(InIndex);
00240 
00241     if (GraphAction.IsValid())
00242     {
00243         TreeView.Pin()->Private_OnItemClicked(GraphAction);
00244     }
00245     else
00246     {
00247         UE_LOG(LogOpenAccessibility, Warning, TEXT("SelectGraphAction: Provided GraphAction is
       Invalid."));
00248     }
00249 }
```

4.46.3.18 SetFilterText()

```
void UAccessibilityAddNodeContextMenu::SetFilterText (
            const FString & InFilterText )
```

Overrides the Current Filter Text with the given string.

Parameters

InFilterText	The String to Override with.

Definition at line 277 of file AccessibilityAddNodeContextMenu.cpp.

```
00278 {
00279     FilterTextBox.Pin()->SetText(FText::FromString(InFilterText));
00280 }
```

4.46.3.19 SetScrollDistance()

```
void UAccessibilityAddNodeContextMenu::SetScrollDistance (
            const float InScrollDistance )
```

Sets the Scroll Distance of the TreeView.

Parameters

InScrollDistance	The Value to Set the Scroll Distance to.

Definition at line 296 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    TreeView.Pin()->SetScrollOffset(InScrollDistance);
}
```

4.46.3.20 SetScrollDistanceBottom()

```
void UAccessibilityAddNodeContextMenu::SetScrollDistanceBottom ( )
```

Sets the Scroll Distance to the Bottom of the TreeView. Taking the View to the Last Item in the TreeView.

Definition at line 317 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    TreeView.Pin()->ScrollToBottom();
}
```

4.46.3.21 SetScrollDistanceTop()

```
void UAccessibilityAddNodeContextMenu::SetScrollDistanceTop ( )
```

Sets the Scroll Distance to the Top of the TreeView. Taking the View to the First Item in the TreeView.

Definition at line 312 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    TreeView.Pin()->ScrollToTop();
}
```

4.46.3.22 Tick()

```
bool UAccessibilityAddNodeContextMenu::Tick (
            float DeltaTime )  [override], [virtual]
```

Reimplemented from UPhraseTreeContextMenuObject.

Definition at line 112 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    if (!GraphMenu.IsValid() || !Menu.IsValid())
        return false;

    if (DoesItemsRequireRefresh())
        RefreshAccessibilityWidgets();

    TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

    // Set Previous Vars For Next Tick
    PrevFilterString = FilterTextBox.Pin()->GetText().ToString();
    PrevNumItemsBeingObserved = TreeViewPtr->GetNumItemsBeingObserved();
    PrevNumGeneratedChildren = TreeViewPtr->GetNumGeneratedChildren();
    PrevScrollDistance = TreeViewPtr->GetScrollDistance().Y;

    return true;
}
```

4.46.3.23 ToggleContextAwareness()

```
void UAccessibilityAddNodeContextMenu::ToggleContextAwareness ( )
```

Toggles the Context Awareness of the Node List.

Definition at line 322 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    ContextAwarenessCheckBox.Pin()->ToggleCheckedState();
}
```

4.46.3.24 UpdateAccessibilityWidget()

```
void UAccessibilityAddNodeContextMenu::UpdateAccessibilityWidget (
            TSharedRef< STableRow< TSharedPtr< FGraphActionNode > > > ItemWidget )  [protected]
```

Updates the previously applied Accessibility Widget, with the new index.

Parameters

ItemWidget	The Item to update.

Definition at line 339 of file AccessibilityAddNodeContextMenu.cpp.

```
{
    TSharedPtr<SContentIndexer> ItemContent =
  StaticCastSharedPtr<SContentIndexer>(ItemWidget->GetContent());

    ItemContent->UpdateIndex(ItemWidget->GetIndexInList());
}
```

4.46.4 Member Data Documentation

4.46.4.1 ContextAwarenessCheckBox

```
TWeakPtr<SCheckBox> UAccessibilityAddNodeContextMenu::ContextAwarenessCheckBox
```

The Context Awareness CheckBox for the Context Menu. Used for toggling Context Awareness, in searching for GraphNodes.

Definition at line 203 of file AccessibilityAddNodeContextMenu.h.

4.46.4.2 FilterTextBox

```
TWeakPtr<SEditableTextBox> UAccessibilityAddNodeContextMenu::FilterTextBox
```

The SEditableTextBox for the Context Menu. Used for Filtering through GraphNodes.

Definition at line 198 of file AccessibilityAddNodeContextMenu.h.

4.46.4.3 GraphMenu

```
TWeakPtr<SGraphActionMenu> UAccessibilityAddNodeContextMenu::GraphMenu
```

The SGraphActionMenu for the Context Menu.

Definition at line 188 of file AccessibilityAddNodeContextMenu.h.

4.46.4.4 PrevFilterString

```
FString UAccessibilityAddNodeContextMenu::PrevFilterString  [protected]
```

Definition at line 207 of file AccessibilityAddNodeContextMenu.h.

4.46.4.5 PrevNumGeneratedChildren

```
int32 UAccessibilityAddNodeContextMenu::PrevNumGeneratedChildren  [protected]
```

Definition at line 209 of file AccessibilityAddNodeContextMenu.h.

4.46.4.6 PrevNumItemsBeingObserved

```
int32 UAccessibilityAddNodeContextMenu::PrevNumItemsBeingObserved  [protected]
```

Definition at line 208 of file AccessibilityAddNodeContextMenu.h.

4.46.4.7 PrevScrollDistance

```
double UAccessibilityAddNodeContextMenu::PrevScrollDistance  [protected]
```

Definition at line 210 of file AccessibilityAddNodeContextMenu.h.

4.46.4.8 TreeView

```
TWeakPtr<STreeView<TSharedPtr<FGraphActionNode> > > UAccessibilityAddNodeContextMenu::Tree↩
View
```

The STreeView for the Context Menu.

Definition at line 193 of file AccessibilityAddNodeContextMenu.h.

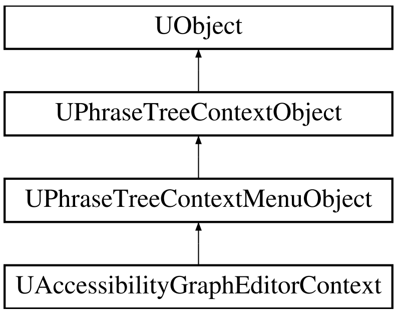
The documentation for this class was generated from the following files:

- Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityAddNodeContextMenu.h
- Source/OpenAccessibility/Private/AccessibilityWrappers/AccessibilityAddNodeContextMenu.cpp

4.47 UAccessibilityGraphEditorContext Class Reference

`#include <AccessibilityGraphEditorContext.h>`

Inheritance diagram for UAccessibilityGraphEditorContext:


UObject
UPhraseTreeContextObject
UPhraseTreeContextMenuObject
UAccessibilityGraphEditorContext

Classes

- struct FTreeViewTickRequirements

Public Member Functions

- virtual void Init (TSharedRef< IMenu > InMenu, TSharedRef< FPhraseNode > InContextRoot) override
- virtual bool Tick (float DeltaTime) override
- virtual bool Close () override
- virtual void ScaleMenu (const float ScaleFactor=1.5f) override
- TSharedPtr< FGraphActionNode > GetTreeViewAction (const int32 &InIndex)
- void SelectAction (const int32 &InIndex)
- FString GetFilterText ()
- void SetFilterText (const FString &NewString)
- void AppendFilterText (const FString &StringToAdd)
- void SetScrollDistance (const float NewDistance)
- void AppendScrollDistance (const float DistanceToAdd)
- void SetScrollDistanceTop ()
- void SetScrollDistanceBottom ()

Protected Member Functions

- const int32 GetStaticIndexOffset ()
- bool FindGraphActionMenu (const TSharedRef< SWidget > &SearchRoot)
- bool FindTreeView (const TSharedRef< SWidget > &SearchRoot)
- bool FindStaticComponents (const TSharedRef< SWidget > &SearchRoot)
- bool TreeViewCanTick ()
- bool TreeViewRequiresTick ()
- void TickTreeViewAccessibility ()
- void UpdateAccessibilityWidget (const TSharedRef< SContentIndexer > &ContextIndexer, const int32 &NewIndex)
- const TSharedRef< SContentIndexer > CreateAccessibilityWrapper (const TSharedRef< SWidget > &ContentToWrap, const int32 &Index)

Protected Attributes

- FTreeViewTickRequirements TreeViewTickRequirements
- TWeakPtr< SGraphActionMenu > GraphMenu = TWeakPtr<SGraphActionMenu>()
- TWeakPtr< SEditableTextBox > FilterTextBox = TWeakPtr<SEditableTextBox>()
- TWeakPtr< STreeView< TSharedPtr< FGraphActionNode > > > TreeView = TWeakPtr<STree↩View<TSharedPtr<FGraphActionNode>>>()
- TArray< TWeakPtr< SCheckBox > > CheckBoxes = TArray<TWeakPtr<SCheckBox>>()

Additional Inherited Members

4.47.1 Detailed Description

A Dynamic Phrase Tree Context Object for Most Node Editor Based Context Menus.

Definition at line 20 of file AccessibilityGraphEditorContext.h.

4.47.2 Constructor & Destructor Documentation

4.47.2.1 UAccessibilityGraphEditorContext()

```
UAccessibilityGraphEditorContext::UAccessibilityGraphEditorContext ( )
```

Definition at line 13 of file AccessibilityGraphEditorContext.cpp.

```
00014     : Super()
00015 {
00016
00017 }
```

4.47.3 Member Function Documentation

4.47.3.1 AppendFilterText()

```
void UAccessibilityGraphEditorContext::AppendFilterText (
             const FString & StringToAdd )
```

Appends the provided string to the Context Menus SearchBar, if it contains one.

Parameters

StringToAdd	The Text to Append to the End of the Active SearchBar.

Definition at line 152 of file AccessibilityGraphEditorContext.cpp.

```
{
    if (!FilterTextBox.IsValid())
        return;

    TSharedPtr<SEditableTextBox> FilterTextBoxPtr = FilterTextBox.Pin();

    FilterTextBoxPtr->SetText(
        FText::FromString( FilterTextBoxPtr->GetText().ToString() + TEXT(" ") + StringToAdd )
    );
}
```

4.47.3.2 AppendScrollDistance()

```
void UAccessibilityGraphEditorContext::AppendScrollDistance (
            const float DistanceToAdd )
```

Adds the Scroll Distance of the Context Menus TreeView, if it contains one.

Parameters

DistanceToAdd	The distance to append to the Scroll Area.

Definition at line 172 of file AccessibilityGraphEditorContext.cpp.

```
{
    auto TreeViewPtr = TreeView.Pin();

    if (TreeViewPtr->GetScrollOffset() + DistanceToAdd < 0.0f)
    {
        TreeViewPtr->SetScrollOffset(0.0f);
        return;
    }

    TreeViewPtr->AddScrollOffset(DistanceToAdd);
}
```

4.47.3.3 Close()

```
bool UAccessibilityGraphEditorContext::Close ( )  [override], [virtual]
```

Closes the Graph Editor Context Wrapper Instance.

Returns

True on successful Closing of the Context Menu, False on Failure.

Reimplemented from UPhraseTreeContextMenuObject.

Definition at line 64 of file AccessibilityGraphEditorContext.cpp.

```
{
    Super::Close();

    return true;
}
```

4.47.3.4 CreateAccessibilityWrapper()

```
const TSharedRef< SContentIndexer > UAccessibilityGraphEditorContext::CreateAccessibility↩
Wrapper (
            const TSharedRef< SWidget > & ContentToWrap,
            const int32 & Index )  [protected]
```

Creates a Content Indexer wrapping the provided Widget.

Parameters

ContentToWrap	The Content to Wrap with an Indexer.
Index	The Index of the Provided Content.

Returns

A Shared Reference of the created Content Indexer, wrapping the provided Content.

Definition at line 349 of file AccessibilityGraphEditorContext.cpp.

```
{
    return SNew(SContentIndexer)
        .IndexValue(Index)
        .IndexPositionToContent(EIndexerPosition::Left)
        .ContentToIndex(ContentToWrap);
}
```

4.47.3.5 FindGraphActionMenu()

```
bool UAccessibilityGraphEditorContext::FindGraphActionMenu (
            const TSharedRef< SWidget > & SearchRoot )  [protected]
```

Finds the SGraphActionMenu Widget descending from the provided widget.

Parameters

SearchRoot	The Starting Point for the Widget Search.

Returns

True if a GraphActionMenu Widget was Found, otherwise False.

Definition at line 200 of file AccessibilityGraphEditorContext.cpp.

```
{
    TSharedPtr<SGraphActionMenu> GraphActionMenu = GetWidgetDescendant<SGraphActionMenu>(SearchRoot,
  TEXT("SGraphActionMenu"));
    if (GraphActionMenu.IsValid())
    {
        GraphMenu = GraphActionMenu;
        FilterTextBox = GraphActionMenu->GetFilterTextBox();

        return true;
    }

    return false;
}
```

4.47.3.6 FindStaticComponents()

```
bool UAccessibilityGraphEditorContext::FindStaticComponents (
            const TSharedRef< SWidget > & SearchRoot )  [protected]
```

Finds any Static Components of the Context Menu and sorts them into the necessary arrays.

Parameters

SearchRoot	The Starting Point for the Widget Search.

Returns

True if Static Components were Found, otherwise False.

Definition at line 230 of file AccessibilityGraphEditorContext.cpp.

```
{
    TArray<FSlotBase*> FoundComponentSlots = GetWidgetSlotsByType(
        SearchRoot,
        TSet<FString> {
            TEXT("SCheckBox")
        }
    );

    if (!FoundComponentSlots.IsEmpty())
    {
        // Sort and Index the Static Components.
        for (int i = 0; i < FoundComponentSlots.Num(); i++)
        {
            FSlotBase* FoundComponentSlot = FoundComponentSlots[i];

            TSharedPtr<SWidget> DetachedWidget = FoundComponentSlot->DetachWidget();
            if (!DetachedWidget.IsValid())
                continue;

            int32 ComponentIndex = -1;
            FString ComponentType = DetachedWidget->GetTypeAsString();

            if (ComponentType == "SCheckBox")
            {
                ComponentIndex = CheckBoxes.Num();
                CheckBoxes.Add(StaticCastSharedPtr<SCheckBox>(DetachedWidget));
            }

            FoundComponentSlot->AttachWidget(
                SNew(SContentIndexer)
                .IndexValue(ComponentIndex)
                .IndexPositionToContent(EIndexerPosition::Left)
                .ContentToIndex(DetachedWidget)
            );
        }

        return true;
    }

    return false;
}
```

4.47.3.7 FindTreeView()

```
bool UAccessibilityGraphEditorContext::FindTreeView (
            const TSharedRef< SWidget > & SearchRoot )  [protected]
```

Finds the STreeView Widget descending from the provided widget.

Parameters

SearchRoot	The Starting Point for the Widget Search.

Returns

True if a TreeView Widget was Found, otherwise False.

Definition at line 214 of file AccessibilityGraphEditorContext.cpp.

```
{
    TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> ContextTreeView =
  GetWidgetDescendant<STreeView<TSharedPtr<FGraphActionNode»>(
        SearchRoot,
        TEXT("STreeView<TSharedPtr<FGraphActionNode»")
    );
    if (ContextTreeView.IsValid())
    {
        TreeView = ContextTreeView;

        return true;
    }

    return false;
}
```

4.47.3.8 GetFilterText()

`FString UAccessibilityGraphEditorContext::GetFilterText ( )`

Gets Filter Text of the Context Menus SearchBar, if it contains one.

Returns

The Current Filter Text of the Context Menus SearchBar, an Empty String on Failure.

Definition at line 137 of file AccessibilityGraphEditorContext.cpp.

```
{
    return FilterTextBox.IsValid() ? FilterTextBox.Pin()->GetText().ToString() : FString();
}
```

4.47.3.9 GetStaticIndexOffset()

`const int32 UAccessibilityGraphEditorContext::GetStaticIndexOffset ( ) [protected]`

Gets the Offset in Indexes of Found Static Components of the Context Menu.

Returns

The Offset of the Static Components Indexes.

Definition at line 195 of file AccessibilityGraphEditorContext.cpp.

```
{
    return CheckBoxes.Num();
}
```

4.47.3.10 GetTreeViewAction()

```
TSharedPtr< FGraphActionNode > UAccessibilityGraphEditorContext::GetTreeViewAction (
            const int32 & InIndex )
```

Gets an Action on the Active TreeView, based on the provided Index.

Parameters

InIndex	The Index of the TreeView Action to Find.

Returns

A Valid Shared Pointer of the Found Action, an Invalid Shared Pointer on Failure.

Definition at line 93 of file AccessibilityGraphEditorContext.cpp.

```
{
    TArrayView<const TSharedPtr<FGraphActionNode>> Items = TreeView.Pin()->GetItems();

    if (TreeView.IsValid() && Items.Num() > InIndex && InIndex >= 0)
        return TreeView.Pin()->GetItems()[InIndex];

    return TSharedPtr<FGraphActionNode>();
}
```

4.47.3.11 Init()

```
void UAccessibilityGraphEditorContext::Init (
            TSharedRef< IMenu > InMenu,
            TSharedRef< FPhraseNode > InContextRoot ) [override], [virtual]
```

Initializes the Graph Editor Context Wrapper.

Parameters

InMenu	The Interface of the Graph Editor Context Menu.
InContextRoot	A Reference to the Originating PhraseNode of this Context Object.

Reimplemented from UPhraseTreeContextMenuObject.

Definition at line 19 of file AccessibilityGraphEditorContext.cpp.

```
{
    Super::Init(InMenu, InContextRoot);

    TSharedRef<SWindow> WindowRef = Window.Pin().ToSharedRef();

    if (!FindGraphActionMenu(WindowRef))
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphEditorContext: Cannot Find a SGraphActionMenu
    Widget"));
    }

    if (!FindStaticComponents(WindowRef))
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphEditorContext: Cannot Find Any Static
    Components"));
    }

    if (!FindTreeView(WindowRef))
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphEditorContext: Cannot Find a STreeView
    Widget"));
    }
    else
    {
        TreeViewTickRequirements = FTreeViewTickRequirements();
    }
}
```

4.47.3.12 ScaleMenu()

```
void UAccessibilityGraphEditorContext::ScaleMenu (
            const float ScaleFactor = 1.5f )  [override], [virtual]
```

Scales Elements of the Context Menu, by the provided Scalar.

Parameters

ScaleFactor	The Scalar to Scale Menu Elements By. (1.5 by Default)

Reimplemented from UPhraseTreeContextMenuObject.

Definition at line 71 of file AccessibilityGraphEditorContext.cpp.

```
{
    Super::ScaleMenu(ScaleFactor);

    // Scale TreeView
    if (TreeView.IsValid())
    {
        TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

        TreeViewPtr->SetItemHeight(16 * ScaleFactor);
    }

    // Scale Window Element
    if (Window.IsValid())
    {
        TSharedPtr<SWindow> WindowPtr = Window.Pin();

        WindowPtr->SetSizingRule(ESizingRule::UserSized);
        WindowPtr->Resize(WindowPtr->GetSizeInScreen() * ScaleFactor);
    }
}
```

4.47.3.13 SelectAction()

```
void UAccessibilityGraphEditorContext::SelectAction (
            const int32 & InIndex )
```

Selects the Action on the Graph Editor Context Menu, based on the given index.

Parameters

InIndex	The Index of the Action To Perform.

Definition at line 103 of file AccessibilityGraphEditorContext.cpp.

```
{
    if (InIndex < 0)
        return;

    if (!CheckBoxes.IsEmpty() && InIndex < CheckBoxes.Num())
    {
        if (CheckBoxes[InIndex].IsValid())
        {
            CheckBoxes[InIndex].Pin()->ToggleCheckedState();
            return;
        }
    }

    TSharedPtr<FGraphActionNode> ChosenTreeViewAction = GetTreeViewAction(InIndex -
  GetStaticIndexOffset());
    if (!ChosenTreeViewAction.IsValid())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("SelectGraphAction: Provided TreeView Action is
    Invalid"))
        return;
    }

    auto TreeViewPtr = TreeView.Pin();
    if (ChosenTreeViewAction->IsActionNode())
    {
        TreeViewPtr->Private_ClearSelection();
        TreeViewPtr->Private_SetItemSelection(ChosenTreeViewAction, true, true);
        TreeViewPtr->Private_SignalSelectionChanged(ESelectInfo::Type::OnMouseClick);
    }
    else
    {
        TreeViewPtr->Private_OnItemDoubleClicked(ChosenTreeViewAction);
    }
}
```

4.47.3.14 SetFilterText()

```
void UAccessibilityGraphEditorContext::SetFilterText (
            const FString & NewString )
```

Sets the Filter Text of the Context Menus SearchBar, if it contains one.

Parameters

NewString	The New Text of the SearchBar.

Definition at line 142 of file AccessibilityGraphEditorContext.cpp.

```
{
    if (!FilterTextBox.IsValid())
        return;

    FilterTextBox.Pin()->SetText(
        FText::FromString(NewString)
    );
}
```

4.47.3.15 SetScrollDistance()

```
void UAccessibilityGraphEditorContext::SetScrollDistance (
            const float NewDistance )
```

Sets the Scroll Distance of the Context Menus TreeView, if it contains one.

Parameters

NewDistance	The New Distance of the Scroll Area.

Definition at line 164 of file AccessibilityGraphEditorContext.cpp.

```
{
    if (TreeView.IsValid())
        return;

    TreeView.Pin()->SetScrollOffset(NewDistance);
}
```

4.47.3.16 SetScrollDistanceBottom()

```
void UAccessibilityGraphEditorContext::SetScrollDistanceBottom ( )
```

Sets the Scroll Distance of the Context Menus TreeView to the Bottom, if it contains one.

Definition at line 190 of file AccessibilityGraphEditorContext.cpp.

```
{
    TreeView.Pin()->ScrollToBottom();
}
```

4.47.3.17 SetScrollDistanceTop()

```
void UAccessibilityGraphEditorContext::SetScrollDistanceTop ( )
```

Sets the Scroll Distance of the Context Menus TreeView to the Top, if it contains one.

Definition at line 185 of file AccessibilityGraphEditorContext.cpp.

```
{
    TreeView.Pin()->ScrollToTop();
}
```

4.47.3.18 Tick()

```
bool UAccessibilityGraphEditorContext::Tick (
             float DeltaTime ) [override], [virtual]
```

Reimplemented from UPhraseTreeContextMenuObject.

Definition at line 45 of file AccessibilityGraphEditorContext.cpp.

```
{
    Super::Tick(DeltaTime);

    if (TreeViewCanTick())
    {
        TickTreeViewAccessibility();

        TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

        TreeViewTickRequirements.PrevSearchText = FilterTextBox.Pin()->GetText().ToString();
        TreeViewTickRequirements.PrevNumGeneratedChildren = TreeViewPtr->GetNumGeneratedChildren();
        TreeViewTickRequirements.PrevNumItemsBeingObserved = TreeViewPtr->GetNumItemsBeingObserved();
        TreeViewTickRequirements.PrevScrollDistance = TreeViewPtr->GetScrollDistance().Y;
    }

    return true;
}
```

4.47.3.19 TickTreeViewAccessibility()

```
void UAccessibilityGraphEditorContext::TickTreeViewAccessibility ( )  [protected]
```

Updates the TreeView Accessibility Components.

Definition at line 297 of file AccessibilityGraphEditorContext.cpp.

```
{
    if (!TreeViewRequiresTick())
        return;

    TSharedPtr<STreeView<TSharedPtr<FGraphActionNode>>> TreeViewPtr = TreeView.Pin();

    TArray<TSharedPtr<FGraphActionNode>> Items = TArray<TSharedPtr<FGraphActionNode>>(
        TreeViewPtr->GetRootItems()
    );


    TSharedPtr<STableRow<TSharedPtr<FGraphActionNode>>> ItemWidget = nullptr;
    const int32 IndexOffset = GetStaticIndexOffset();

    while (Items.Num() > 0)
    {
        const TSharedPtr<FGraphActionNode> Item = Items[0];
        Items.RemoveAt(0);

        if (TreeViewPtr->IsItemExpanded(Item))
            Items.Append(Item->Children);

        ItemWidget = StaticCastSharedPtr<STableRow<TSharedPtr<FGraphActionNode>>>(
            TreeViewPtr->WidgetFromItem(Item)
        );
        if (!ItemWidget.IsValid())
            continue;

        TSharedPtr<SWidget> ItemContent = ItemWidget->GetContent();

        if (ItemContent->GetType() == "SContentIndexer")
        {
            UpdateAccessibilityWidget(
                StaticCastSharedRef<SContentIndexer>(ItemContent.ToSharedRef()),
                IndexOffset + ItemWidget->GetIndexInList()
            );
        }
        else
        {
            ItemWidget->SetContent(
                CreateAccessibilityWrapper(ItemContent.ToSharedRef(), IndexOffset +
       ItemWidget->GetIndexInList())
            );
        }
    }
}
```

4.47.3.20 TreeViewCanTick()

```
bool UAccessibilityGraphEditorContext::TreeViewCanTick ( )  [protected]
```

Checks if all required components for ticking the TreeView are available.

Returns

True if all required components are found for TreeView Ticking, otherwise False.

Definition at line 273 of file AccessibilityGraphEditorContext.cpp.

```
{
    return TreeView.IsValid() && GraphMenu.IsValid();
}
```

4.47.3.21 TreeViewRequiresTick()

```
bool UAccessibilityGraphEditorContext::TreeViewRequiresTick ( ) [protected]
```

Checks if the Dynamic TreeView Accessibility Components Require a Refresh.

Returns

True if the TreeView Accessibility Assets Require a Refresh.

Definition at line 278 of file AccessibilityGraphEditorContext.cpp.

```
{
    if (!TreeView.IsValid() || !GraphMenu.IsValid())
        return false;

    bool bFilterTextChange = FilterTextBox.IsValid()
        ? FilterTextBox.Pin()->GetText().ToString() != TreeViewTickRequirements.PrevSearchText
        : false;

    TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

    return (
        bFilterTextChange ||
        TreeViewPtr->GetNumItemsBeingObserved() != TreeViewTickRequirements.PrevNumItemsBeingObserved ||
        TreeViewPtr->GetNumGeneratedChildren() != TreeViewTickRequirements.PrevNumGeneratedChildren ||
        TreeViewPtr->GetScrollDistance().Y != TreeViewTickRequirements.PrevScrollDistance
    );
}
```

4.47.3.22 UpdateAccessibilityWidget()

```
void UAccessibilityGraphEditorContext::UpdateAccessibilityWidget (
            const TSharedRef< SContentIndexer > & ContextIndexer,
            const int32 & NewIndex ) [protected]
```

Updates the provided Content Indexer Widget with the given Index.

Parameters

ContextIndexer	The Context Indexer Widget to Update.
NewIndex	The Index to update the Context Indexer With.

Definition at line 344 of file AccessibilityGraphEditorContext.cpp.

```
{
    ContentIndexer->UpdateIndex(NewIndex);
}
```

4.47.4 Member Data Documentation

4.47.4.1 CheckBoxes

```
TArray<TWeakPtr<SCheckBox> > UAccessibilityGraphEditorContext::CheckBoxes = TArray<TWeak←
Ptr<SCheckBox>>() [protected]
```

Definition at line 206 of file AccessibilityGraphEditorContext.h.

4.47.4.2 FilterTextBox

```
TWeakPtr<SEditableTextBox> UAccessibilityGraphEditorContext::FilterTextBox = TWeakPtr<SEditable←
TextBox>()  [protected]
```

Definition at line 202 of file AccessibilityGraphEditorContext.h.

4.47.4.3 GraphMenu

```
TWeakPtr<SGraphActionMenu> UAccessibilityGraphEditorContext::GraphMenu = TWeakPtr<SGraph←
ActionMenu>()  [protected]
```

Definition at line 201 of file AccessibilityGraphEditorContext.h.

4.47.4.4 TreeView

```
TWeakPtr<STreeView<TSharedPtr<FGraphActionNode> > > UAccessibilityGraphEditorContext::Tree←
View = TWeakPtr<STreeView<TSharedPtr<FGraphActionNode>>>()  [protected]
```

Definition at line 204 of file AccessibilityGraphEditorContext.h.

4.47.4.5 TreeViewTickRequirements

```
FTreeViewTickRequirements UAccessibilityGraphEditorContext::TreeViewTickRequirements  [protected]
```

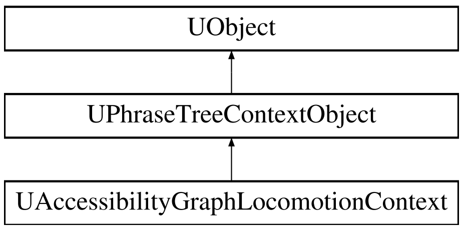
Definition at line 199 of file AccessibilityGraphEditorContext.h.

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

- Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityGraphEditorContext.h
- Source/OpenAccessibility/Private/AccessibilityWrappers/AccessibilityGraphEditorContext.cpp

4.48 UAccessibilityGraphLocomotionContext Class Reference

Inheritance diagram for UAccessibilityGraphLocomotionContext:



Public Member Functions

- UAccessibilityGraphLocomotionContext (const FObjectInitializer &ObjectInitializer)
- void Init ()
- void Init (TSharedRef< SGraphEditor > InGraphEditor)
- bool SelectChunk (const int32 &Index)
- bool RevertToPreviousView ()
- void ConfirmSelection ()
- void CancelLocomotion ()
- virtual bool Close () override

Protected Member Functions

- bool MoveViewport (const FVector2D &InTopLeft, const FVector2D &InBottomRight) const
- bool MoveViewport (const FPanelViewPosition &NewViewPosition) const
- void ChangeChunkVis (const int32 &Index, const FLinearColor &NewColor=FLinearColor::Yellow)
- void CreateVisualGrid (const TSharedRef< SGraphEditor > InGraphEditor)
- void GenerateVisualChunks (const TSharedRef< SGraphEditor > InGraphEditor, FIntVector2 InVisual↩ ChunkSize=FIntVector2(10))
- void CalculateVisualChunksBounds ()
- void RemoveVisualGrid ()
- void HideNativeVisuals ()
- void UnHideNativeVisuals ()
- void OnFocusChanged (const FFocusEvent &FocusEvent, const FWeakWidgetPath &OldFocusedWidget↩ Path, const TSharedPtr< SWidget > &OldFocusedWidget, const FWidgetPath &NewFocusedWidgetPath, const TSharedPtr< SWidget > &NewFocusedWidget)
- void BindFocusChangedEvent ()
- void UnbindFocusChangedEvent ()

Protected Attributes

- FVector2D StartViewPosition
- float StartViewZoom
- FPanelViewPosition CurrentViewPosition
- TArray< FPanelViewPosition > PreviousPositions
- TArray< FGraphLocomotionChunk > ChunkArray
- FIntVector2 ChunkSize
- TWeakPtr< SUniformGridPanel > GridContainer
- TWeakPtr< SOverlay > GridParent
- TWeakPtr< SGraphEditor > LinkedEditor

4.48.1 Detailed Description

Definition at line 99 of file AccessibilityGraphLocomotionContext.h.

4.48.2 Constructor & Destructor Documentation

4.48.2.1 UAccessibilityGraphLocomotionContext()

```
UAccessibilityGraphLocomotionContext::UAccessibilityGraphLocomotionContext (
              const FObjectInitializer & ObjectInitializer )
```

Definition at line 9 of file AccessibilityGraphLocomotionContext.cpp.

```
    : UPhraseTreeContextObject()
{
    LinkedEditor = TWeakPtr<SGraphEditor>();
}
```

4.48.2.2 ∼UAccessibilityGraphLocomotionContext()

```
UAccessibilityGraphLocomotionContext::∼UAccessibilityGraphLocomotionContext ( )  [virtual]
```

Definition at line 15 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    Close();

    UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: CONTEXT DESTROYED."));
}
```

4.48.3 Member Function Documentation

4.48.3.1 BindFocusChangedEvent()

```
void UAccessibilityGraphLocomotionContext::BindFocusChangedEvent ( )  [protected]
```

Definition at line 364 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    FocusChangedHandle = FSlateApplication::Get().OnFocusChanging()
        .AddUObject(this, &UAccessibilityGraphLocomotionContext::OnFocusChanged);
}
```

4.48.3.2 CalculateVisualChunksBounds()

```
void UAccessibilityGraphLocomotionContext::CalculateVisualChunksBounds ( )  [protected]
```

Definition at line 248 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    if (!LinkedEditor.IsValid())
        return;

    SGraphPanel* LinkedPanel = LinkedEditor.Pin()->GetGraphPanel();
    FVector2D PanelGeoSize = LinkedPanel->GetTickSpaceGeometry().GetLocalSize();

    double ChunkWidgetSizeX = PanelGeoSize.X / ChunkSize.X;
    double ChunkWidgetSizeY = PanelGeoSize.Y / ChunkSize.Y;

    FGraphLocomotionChunk Chunk;
    double ChunkX, ChunkY;

    int32 ArrIndex;
    for (int Y = 0; Y < ChunkSize.Y; Y++)
    {
        for (int X = 0; X < ChunkSize.X; X++)
        {
            ArrIndex = (Y * ChunkSize.X) + X;

            Chunk = ChunkArray[ArrIndex];

            ChunkX = X * ChunkWidgetSizeX;
            ChunkY = Y * ChunkWidgetSizeY;

            Chunk.SetChunkBounds(
                FVector2D(ChunkX, ChunkY),
                FVector2D(ChunkWidgetSizeX + ChunkX, ChunkWidgetSizeY + ChunkY)
            );

            ChunkArray[ArrIndex] = Chunk;
        }
    }
}
```

4.48.3.3 CancelLocomotion()

`void UAccessibilityGraphLocomotionContext::CancelLocomotion ( )`

Definition at line 121 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    if (LinkedEditor.IsValid())
    {
        LinkedEditor.Pin()->SetViewLocation(StartViewPosition, StartViewZoom);

        Close();
    }
}
```

4.48.3.4 ChangeChunkVis()

```
void UAccessibilityGraphLocomotionContext::ChangeChunkVis (
            const int32 & Index,
            const FLinearColor & NewColor = FLinearColor::Yellow )  [protected]
```

Definition at line 172 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    check(Index < ChunkArray.Num() && Index >= 0)

    ChunkArray[Index].SetVisColor(NewColor);
}
```

4.48.3.5 Close()

`bool UAccessibilityGraphLocomotionContext::Close ( )  [override], [virtual]`

Reimplemented from UPhraseTreeContextObject.

Definition at line 131 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    UnbindFocusChangedEvent();

    if (SelectionTimerHandle.IsValid())
        GEditor->GetTimerManager()->ClearTimer(SelectionTimerHandle);

    RemoveVisualGrid();
    UnHideNativeVisuals();

    bIsActive = false;

    RemoveFromRoot();
    MarkAsGarbage();

    UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: CONTEXT CLOSED."));

    return true;
}
```

4.48.3.6 ConfirmSelection()

`void UAccessibilityGraphLocomotionContext::ConfirmSelection ( )`

Definition at line 116 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    Close();
}
```

4.48.3.7 CreateVisualGrid()

```
void UAccessibilityGraphLocomotionContext::CreateVisualGrid (
            const TSharedRef< SGraphEditor > InGraphEditor )  [protected]
```

Definition at line 179 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    TSharedPtr<SOverlay> GraphViewport =
  StaticCastSharedPtr<SOverlay>(InGraphEditor->GetGraphPanel()->GetParentWidget());
    if (!GraphViewport.IsValid())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: NO GRAPH VIEWPORT FOUND."));
        return;
    }

    GridParent = GraphViewport;

    GraphViewport->AddSlot()
    .ZOrder(1)
    .VAlign(VAlign_Fill)
    .HAlign(HAlign_Fill)
    [
        SAssignNew(GridContainer, SUniformGridPanel)
    ];
}
```

4.48.3.8 GenerateVisualChunks()

```
void UAccessibilityGraphLocomotionContext::GenerateVisualChunks (
            const TSharedRef< SGraphEditor > InGraphEditor,
            FIntVector2 InVisualChunkSize = FIntVector2(10) )  [protected]
```

Definition at line 199 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    ChunkArray.Reset(InVisualChunkSize.X * InVisualChunkSize.Y);
    ChunkSize = InVisualChunkSize;

    TSharedPtr<SUniformGridPanel> GridContainerPtr = GridContainer.Pin();

    int32 ChunkIndex = -1;
    TSharedPtr<SBox> ChunkWidget;
    TSharedPtr<SBorder> ChunkVisWidget;
    TSharedPtr<SIndexer> ChunkIndexer;

    for (int32 Y = 0; Y < InVisualChunkSize.Y; Y++)
    {
        for (int32 X = 0; X < InVisualChunkSize.X; X++)
        {
            ChunkIndex = X + (Y * InVisualChunkSize.X);
            FGraphLocomotionChunk& GraphChunk = ChunkArray.EmplaceAt_GetRef(ChunkIndex);

            GridContainerPtr->AddSlot(X, Y)
            [
                SAssignNew(ChunkWidget, SBox)
                [
                    SAssignNew(ChunkVisWidget, SBorder)
                    .Padding(0.5f)
                    .BorderBackgroundColor(FLinearColor::Yellow)
                    [
                        SNew(SBorder)
                        .HAlign(HAlign_Center)
                        .VAlign(VAlign_Center)
                        .BorderBackgroundColor(FLinearColor::Yellow)
                        [
                            SAssignNew(ChunkIndexer, SIndexer)
                            .TextColor(FLinearColor::Yellow)
                            .IndexValue(ChunkIndex)
                        ]
                    ]
                ]
            ];

            GraphChunk.ChunkWidget = ChunkWidget;
            GraphChunk.ChunkVisWidget = ChunkVisWidget;
            GraphChunk.ChunkIndexer = ChunkIndexer;
        }
    }

    CalculateVisualChunksBounds();
}
```

4.48.3.9 HideNativeVisuals()

`void UAccessibilityGraphLocomotionContext::HideNativeVisuals ( ) [protected]`

Definition at line 302 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    NativeWidgetVisibility.Empty();

    TSharedPtr<SOverlay> GraphViewport = GridParent.Pin();
    TSharedPtr<SUniformGridPanel> VisualGrid = GridContainer.Pin();
    SGraphPanel* GraphPanel = LinkedEditor.Pin()->GetGraphPanel();

    FChildren* ViewportChildren = GraphViewport->GetChildren();

    TSharedPtr<SWidget> ChildWidget;
    for (int32 i = 0; i < ViewportChildren->Num(); i++)
    {
        ChildWidget = ViewportChildren->GetChildAt(i);

        if (ChildWidget != VisualGrid && ChildWidget.Get() != GraphPanel)
        {
            NativeWidgetVisibility.Add(ChildWidget.Get(), ChildWidget->GetVisibility());

            ChildWidget->SetVisibility(EVisibility::Hidden);
        }
    }
}
```

4.48.3.10 Init() [1/2]

```
void UAccessibilityGraphLocomotionContext::Init ( )
```

Definition at line 22 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    {
        TSharedPtr<SDockTab> ActiveTab = FGlobalTabmanager::Get()->GetActiveTab();
        if (!ActiveTab.IsValid())
        {
            UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: NO ACTIVE TAB FOUND."));
            return;
        }

        LinkedEditor = StaticCastSharedRef<SGraphEditor>(ActiveTab->GetContent());
        if (!LinkedEditor.IsValid())
        {
            UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: CURRENT ACTIVE TAB IS NOT OF
       TYPE - SGraphEditor"));
            return;
        }
    }

    TSharedPtr<SGraphEditor> LinkedEditorPtr = LinkedEditor.Pin();

    Init(LinkedEditorPtr.ToSharedRef());
}
```

4.48.3.11 Init() [2/2]

```
void UAccessibilityGraphLocomotionContext::Init (
            TSharedRef< SGraphEditor > InGraphEditor )
```

Definition at line 45 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    LinkedEditor = InGraphEditor;

    InGraphEditor->GetViewLocation(StartViewPosition, StartViewZoom);
    InGraphEditor->ZoomToFit(false);

    CreateVisualGrid(InGraphEditor);
    GenerateVisualChunks(InGraphEditor, FIntVector2(6, 4));

    HideNativeVisuals();

    BindFocusChangedEvent();
}
```

4.48.3.12 MoveViewport() [1/2]

```
bool UAccessibilityGraphLocomotionContext::MoveViewport (
            const FPanelViewPosition & NewViewPosition ) const [protected]
```

Definition at line 162 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    if (!LinkedEditor.IsValid())
        return false;

    SGraphPanel* LinkedPanel = LinkedEditor.Pin()->GetGraphPanel();

    return LinkedPanel->JumpToRect(NewViewPosition.TopLeft, NewViewPosition.BotRight);
}
```

4.48.3.13 MoveViewport() [2/2]

```
bool UAccessibilityGraphLocomotionContext::MoveViewport (
            const FVector2D & InTopLeft,
            const FVector2D & InBottomRight ) const  [protected]
```

Definition at line 151 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    if (!LinkedEditor.IsValid())
        return false;

    TSharedPtr<SGraphEditor> LinkedEditorPtr = LinkedEditor.Pin();
    SGraphPanel* LinkedPanel = LinkedEditorPtr->GetGraphPanel();

    return LinkedPanel->JumpToRect(InTopLeft, InBottomRight);
}
```

4.48.3.14 OnFocusChanged()

```
void UAccessibilityGraphLocomotionContext::OnFocusChanged (
            const FFocusEvent & FocusEvent,
            const FWeakWidgetPath & OldFocusedWidgetPath,
            const TSharedPtr< SWidget > & OldFocusedWidget,
            const FWidgetPath & NewFocusedWidgetPath,
            const TSharedPtr< SWidget > & NewFocusedWidget )  [protected]
```

Definition at line 344 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    if (!bIsActive)
        return;

    UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: FOCUS CHANGED."));

    TSharedPtr<SGraphEditor> LinkedEditorPtr = LinkedEditor.Pin();

    if (!NewFocusedWidgetPath.ContainsWidget(LinkedEditorPtr.ToSharedRef()))
    {
        bIsActive = false;
        Close();
    }
}
```

4.48.3.15 RemoveVisualGrid()

```
void UAccessibilityGraphLocomotionContext::RemoveVisualGrid ( )  [protected]
```

Definition at line 284 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    TSharedPtr<SUniformGridPanel> GridContainerPtr = GridContainer.Pin();
    if (GridContainerPtr.IsValid())
    {
        TSharedPtr<SOverlay> ParentWidget = StaticCastSharedPtr<SOverlay>(
            GridContainerPtr->GetParentWidget()
        );

        if (ParentWidget.IsValid()) {
            ParentWidget->RemoveSlot(GridContainerPtr.ToSharedRef());

            GridParent = ParentWidget;
        }
        else UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: PARENT WIDGET NOT FOUND,
    CANNOT REMOVE LOCOMOTION WIDGETS."))
    }
}
```

4.48.3.16 RevertToPreviousView()

```
bool UAccessibilityGraphLocomotionContext::RevertToPreviousView ( )
```

Definition at line 100 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    if (PreviousPositions.IsEmpty())
    {
        LinkedEditor.Pin()->ZoomToFit(false);
        return true;
    }

    if (!MoveViewport(PreviousPositions.Pop()))
    {
        return false;
    }

    return true;
}
```

4.48.3.17 SelectChunk()

```
bool UAccessibilityGraphLocomotionContext::SelectChunk (
            const int32 & Index )
```

Definition at line 60 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    if (Index > ChunkArray.Num() || Index < 0)
        return false;

    const FGraphLocomotionChunk SelectedChunk = ChunkArray[Index];

    const SGraphPanel* LinkedPanel = LinkedEditor.Pin()->GetGraphPanel();

    const FVector2D GraphTopLeftCoord =
LinkedPanel->PanelCoordToGraphCoord(SelectedChunk.GetChunkTopLeft());
    const FVector2D GraphBottomRightCoord =
LinkedPanel->PanelCoordToGraphCoord(SelectedChunk.GetChunkBottomRight());

    ChangeChunkVis(Index, FLinearColor::Red);

    GEditor->GetTimerManager()->SetTimer(
        SelectionTimerHandle,
        [this, Index, GraphTopLeftCoord, GraphBottomRightCoord]()
        {
            ChangeChunkVis(Index);

            if (MoveViewport(GraphTopLeftCoord, GraphBottomRightCoord))
            {
                if (CurrentViewPosition != FVector2D::ZeroVector)
                    PreviousPositions.Push(CurrentViewPosition);

                CurrentViewPosition = FPanelViewPosition(GraphTopLeftCoord, GraphBottomRightCoord);
            }
            else
            {
                UE_LOG(LogOpenAccessibility, Log, TEXT("Failed To Jump To Viewport Coords (TopLeft: %s
| BottomRight: %s)"),
                    *GraphTopLeftCoord.ToString(), *GraphBottomRightCoord.ToString());
            }
        },
        0.5f,
        false
    );

    return true;
}
```

4.48.3.18 UnbindFocusChangedEvent()

```
void UAccessibilityGraphLocomotionContext::UnbindFocusChangedEvent ( )  [protected]
```

Definition at line 370 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    if (FocusChangedHandle.IsValid())
    {
        FSlateApplication::Get().OnFocusChanging().Remove(FocusChangedHandle);
    }
}
```

4.48.3.19 UnHideNativeVisuals()

```
void UAccessibilityGraphLocomotionContext::UnHideNativeVisuals ( )  [protected]
```

Definition at line 326 of file AccessibilityGraphLocomotionContext.cpp.

```
{
    FChildren* ViewportChildren = GridParent.Pin()->GetChildren();

    TSharedPtr<SWidget> ChildWidget;
    for (int32 i = 0; i < ViewportChildren->Num(); i++)
    {
        ChildWidget = ViewportChildren->GetChildAt(i);

        if (NativeWidgetVisibility.Contains(ChildWidget.Get()))
        {
            ChildWidget->SetVisibility(NativeWidgetVisibility[ChildWidget.Get()]);
        }
    }

    NativeWidgetVisibility.Empty();
}
```

4.48.4 Member Data Documentation

4.48.4.1 ChunkArray

```
TArray<FGraphLocomotionChunk> UAccessibilityGraphLocomotionContext::ChunkArray  [protected]
```

Definition at line 160 of file AccessibilityGraphLocomotionContext.h.

4.48.4.2 ChunkSize

```
FIntVector2 UAccessibilityGraphLocomotionContext::ChunkSize  [protected]
```

Definition at line 162 of file AccessibilityGraphLocomotionContext.h.

4.48.4.3 CurrentViewPosition

```
FPanelViewPosition UAccessibilityGraphLocomotionContext::CurrentViewPosition  [protected]
```

Definition at line 155 of file AccessibilityGraphLocomotionContext.h.

4.48.4.4 GridContainer

```
TWeakPtr<SUniformGridPanel> UAccessibilityGraphLocomotionContext::GridContainer  [protected]
```

Definition at line 167 of file AccessibilityGraphLocomotionContext.h.

4.48.4.5 GridParent

```
TWeakPtr<SOverlay> UAccessibilityGraphLocomotionContext::GridParent  [protected]
```

Definition at line 169 of file AccessibilityGraphLocomotionContext.h.

4.48.4.6 LinkedEditor

```
TWeakPtr<SGraphEditor> UAccessibilityGraphLocomotionContext::LinkedEditor  [protected]
```

Definition at line 171 of file AccessibilityGraphLocomotionContext.h.

4.48.4.7 PreviousPositions

```
TArray<FPanelViewPosition> UAccessibilityGraphLocomotionContext::PreviousPositions  [protected]
```

Definition at line 156 of file AccessibilityGraphLocomotionContext.h.

4.48.4.8 StartViewPosition

```
FVector2D UAccessibilityGraphLocomotionContext::StartViewPosition  [protected]
```

Definition at line 153 of file AccessibilityGraphLocomotionContext.h.

4.48.4.9 StartViewZoom

```
float UAccessibilityGraphLocomotionContext::StartViewZoom  [protected]
```

Definition at line 153 of file AccessibilityGraphLocomotionContext.h.

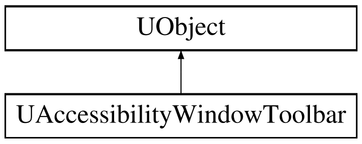
The documentation for this class was generated from the following files:

- Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityGraphLocomotionContext.h
- Source/OpenAccessibility/Private/AccessibilityWrappers/AccessibilityGraphLocomotionContext.cpp

4.49 UAccessibilityWindowToolbar Class Reference

```
#include <AccessibilityWindowToolbar.h>
```

Inheritance diagram for UAccessibilityWindowToolbar:



Public Member Functions

- bool Tick (float DeltaTime)
- void SelectToolbarItem (int32 Index)
- bool IsActiveToolbar (const TSharedRef< SWidget > &ToolkitWidget)
- TSharedPtr< SWidget > GetActiveToolkitWidget () const

4.49.1 Detailed Description

Accessibility Wrapper for Window ToolBar Elements.

Definition at line 15 of file AccessibilityWindowToolbar.h.

4.49.2 Constructor & Destructor Documentation

4.49.2.1 UAccessibilityWindowToolbar()

```
UAccessibilityWindowToolbar::UAccessibilityWindowToolbar ( )
```

Definition at line 9 of file AccessibilityWindowToolbar.cpp.

```
                                                              : UObject()
{
    LastToolkit = TWeakPtr<SWidget>();
    LastTopWindow = TWeakPtr<SWindow>();
    LastToolkitParent = TWeakPtr<SBorder>();

    ConsoleCommands.Add(IConsoleManager::Get().RegisterConsoleCommand(
        TEXT("OpenAccessibiliy.ToolBar.ShowIndexerStats"),
        TEXT("Displays the Indexer Stats for the Toolbar."),

        FConsoleCommandDelegate::CreateLambda([this]() {
            UE_LOG(LogOpenAccessibility, Display, TEXT("| ToolBar Indexer Stats | Indexed Amount: %d |
"), ToolbarIndex.Num())
        })
    ));

    BindTicker();
}
```

4.49.2.2 ∼UAccessibilityWindowToolbar()

```
UAccessibilityWindowToolbar::∼UAccessibilityWindowToolbar ( ) [virtual]
```

Definition at line 27 of file AccessibilityWindowToolbar.cpp.

```
{
    UE_LOG(LogOpenAccessibility, Log, TEXT("AccessibilityToolBar: Destroyed."));

    UnbindTicker();
}
```

4.49.3 Member Function Documentation

4.49.3.1 GetActiveToolkitWidget()

```
TSharedPtr< SWidget > UAccessibilityWindowToolbar::GetActiveToolkitWidget ( ) const
```

Gets the Stored Active Toolkit Widget.

Returns

Shared Pointer to the Active Toolkit Widget, otherwise Invalid Pointer.

Definition at line 240 of file AccessibilityWindowToolbar.cpp.

```
{
    if (LastToolkit.IsValid())
        return LastToolkit.Pin();

    return TSharedPtr<SWidget>();
}
```

4.49.3.2 IsActiveToolbar()

```
bool UAccessibilityWindowToolbar::IsActiveToolbar (
            const TSharedRef< SWidget > & ToolkitWidget )
```

Checks to see if the Active Toolkit being Indexed is the provided Toolkit Widget.

Parameters

ToolkitWidget	Toolkit Widget to Check if it is the active toolkit being Indexed.

Returns

True if the provided toolkit is the active widget, otherwise False.

Definition at line 233 of file AccessibilityWindowToolbar.cpp.

```
{
    return LastToolkit.IsValid()
        ? LastToolkit.Pin() == ToolkitWidget
        : false;
}
```

4.49.3.3 SelectToolbarItem()

```
void UAccessibilityWindowToolbar::SelectToolbarItem (
            int32 Index )
```

Selects the Active ToolBars Element, based on the provided Index.

Parameters

Index	The Index of the ToolBar Element To Select.

Definition at line 197 of file AccessibilityWindowToolbar.cpp.

```
{
    if (ToolbarIndex.IsEmpty())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("ToolBar Index is Empty."))
        return;
    }

    SMultiBlockBaseWidget* LinkedButton;
    if (!ToolbarIndex.GetValue(Index, LinkedButton))
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided Index is Not Linked to a ToolBar Button."))
        return;
    }

    TSharedPtr<const FMultiBlock> MultiBlock = LinkedButton->GetBlock();
    if (!MultiBlock.IsValid())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided ToolBar MultiBlock is Not Valid."))
    }

    TSharedPtr<const FUICommandList> ActionList = MultiBlock->GetActionList();
    TSharedPtr<const FUICommandInfo> Action = MultiBlock->GetAction();

    if (ActionList.IsValid() && Action.IsValid())
    {
        ActionList->ExecuteAction( Action.ToSharedRef() );
    }
    else
    {
        const FUIAction& DirectAction = MultiBlock->GetDirectActions();

        DirectAction.Execute();
    }
}
```

4.49.3.4 Tick()

```
bool UAccessibilityWindowToolbar::Tick (
            float DeltaTime )
```

Definition at line 34 of file AccessibilityWindowToolbar.cpp.

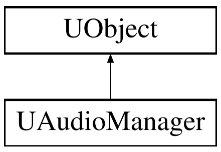
```
{
    TSharedPtr<SWindow> TopWindow = FSlateApplication::Get().GetActiveTopLevelRegularWindow();
    if (!TopWindow.IsValid())
    {
        return true;
    }

    TSharedPtr<SBorder> ContentContainer;
    if (TopWindow != LastTopWindow)
        ContentContainer = GetWindowContentContainer(TopWindow.ToSharedRef());
    else ContentContainer = LastToolkitParent.Pin();

    if (!ContentContainer.IsValid())
    {
        return true;
    }


    TSharedPtr<SWidget> Toolkit = ContentContainer->GetContent();
    if (!Toolkit.IsValid())
    {
        return true;
    }

    if (ApplyToolbarIndexing(Toolkit.ToSharedRef(), TopWindow.ToSharedRef()))
    {
        LastToolkit = Toolkit;
        //UE_LOG(LogOpenAccessibility, Log, TEXT("AccessibilityToolBar: Toolkit Indexing Applied To %s"), *Toolkit->GetTypeAsString());
    }

    LastTopWindow = TopWindow;
    LastToolkitParent = ContentContainer;

    return true;
}
```

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

- Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityWindowToolbar.h
- Source/OpenAccessibility/Private/AccessibilityWrappers/AccessibilityWindowToolbar.cpp

4.50 UAudioManager Class Reference

Inheritance diagram for UAudioManager:



Public Member Functions

- void StartCapturingAudio ()

  Starts The Capturing of Audio onto the Buffer.
- void StopCapturingAudio ()

  Stops the Capturing of Audio onto the Buffer, and sends the audio to the transcription service.

- void PRIVATE_OnAudioGenerate (const float ∗InAudio, int32 NumSamples)

  Callback For When Audio is Generated by The Audio Stream.
- void SaveAudioBufferToWAV (const FString &FilePath)

  Saves the Audio Buffer to a WAV File.
- bool IsCapturingAudio () const

  Is the Audio Manager Currently Capturing Audio.
- int32 GetAudioCaptureSampleRate () const

  Gets the Sample Rate of the Audio Capture.
- int32 GetAudioCaptureNumChannels () const

  Gets the Number of Channels of the Audio Capture.
- void OnDefaultDeviceChanged (EAudioDeviceChangedRole ChangedRole, FString DeviceID)

  Callback for when the Default Audio Device Changes. Allowing for dynamic re-registration of the Audio Generator, to make sure the new device is being used.

Public Attributes

- FAudioManagerSettings Settings

  The Settings of the Audio Manager.
- TDelegate< void(const TArray< float >)> OnAudioReadyForTranscription

  Delegate for when the AudioBuffer is Ready To Be Sent For Transcription.

4.50.1 Detailed Description

Definition at line 50 of file AudioManager.h.

4.50.2 Constructor & Destructor Documentation

4.50.2.1 UAudioManager()

```
UAudioManager::UAudioManager ( )
```

Definition at line 12 of file AudioManager.cpp.

```
{
    Settings = FAudioManagerSettings();

    // Create Audio Capture Object and Initialize Audio Stream
    bIsCapturingAudio = false;
    AudioCapture = NewObject<UAudioCapture>();
    AudioCapture->OpenDefaultAudioStream();
    AudioCapture->StartCapturingAudio();

    RegisterAudioGenerator();

    // Create FileIO Objects
    FileWriter = new Audio::FSoundWavePCMWriter();
}
```

4.50.2.2 ∼UAudioManager()

```
UAudioManager::∼UAudioManager ( )  [virtual]
```

Definition at line 28 of file AudioManager.cpp.

```
{
    UnregisterAudioGenerator();

    AudioCapture->StopCapturingAudio();
    AudioCapture->RemoveFromRoot();

    delete AudioCapture; AudioCapture = nullptr;
    delete FileWriter; FileWriter = nullptr;
}
```

4.50.3 Member Function Documentation

4.50.3.1 GetAudioCaptureNumChannels()

```
int32 UAudioManager::GetAudioCaptureNumChannels ( ) const  [inline]
```

Gets the Number of Channels of the Audio Capture.

Returns

The Number of Channels used in the Audiocapture.

Definition at line 97 of file AudioManager.h.

```
{ return AudioCapture->GetNumChannels(); }
```

4.50.3.2 GetAudioCaptureSampleRate()

```
int32 UAudioManager::GetAudioCaptureSampleRate ( ) const  [inline]
```

Gets the Sample Rate of the Audio Capture.

Returns

The Sample Rate of the Audiocapture.

Definition at line 91 of file AudioManager.h.

```
{ return AudioCapture->GetSampleRate(); }
```

4.50.3.3 IsCapturingAudio()

```
bool UAudioManager::IsCapturingAudio ( ) const  [inline]
```

Is the Audio Manager Currently Capturing Audio.

Returns

True, if Audio is being Captured. False, if Audio is being ignored.

Definition at line 85 of file AudioManager.h.

```
00085 { return bIsCapturingAudio; }
```

4.50.3.4 OnDefaultDeviceChanged()

```
void UAudioManager::OnDefaultDeviceChanged (
            EAudioDeviceChangedRole ChangedRole,
            FString DeviceID )
```

Callback for when the Default Audio Device Changes. Allowing for dynamic re-registration of the Audio Generator, to make sure the new device is being used.

Parameters

ChangedRole	
DeviceID	

Definition at line 88 of file AudioManager.cpp.

```
00089 {
00090     UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Default Device Changed || Role: %d || DeviceID: %s
      ||"), ChangedRole, *DeviceID);
00091
00092     this->UnregisterAudioGenerator();
00093     this->RegisterAudioGenerator();
00094 }
```

4.50.3.5 PRIVATE_OnAudioGenerate()

```
void UAudioManager::PRIVATE_OnAudioGenerate (
            const float * InAudio,
            int32 NumSamples )
```

Callback For When Audio is Generated by The Audio Stream.

Parameters

InAudio	- The Incoming Audiobuffer Array.
NumSamples	- The Size of the Incoming Audiobuffer in Samples.

Definition at line 67 of file AudioManager.cpp.

```
{
    if (bIsCapturingAudio == false)
        return;

    // Need to Check Samples are above threshold and ignore if their run length is too long.

    AudioBuffer.Append(InAudio, NumSamples);
}
```

4.50.3.6 SaveAudioBufferToWAV()

```
void UAudioManager::SaveAudioBufferToWAV (
              const FString & FilePath )
```

Saves the Audio Buffer to a WAV File.

Parameters

FilePath	- The Path To Save the Audiobuffers WAV File.

Definition at line 77 of file AudioManager.cpp.

```
{
    UE_LOG(LogOpenAccessibilityCom, Log, TEXT("Starting to Save Audio Buffer to WAV"));

    Audio::FSampleBuffer SampleBuffer = Audio::FSampleBuffer(AudioBuffer.GetData(), AudioBuffer.Num(),
       AudioCapture->GetNumChannels(), AudioCapture->GetSampleRate());

    FileWriter->BeginWriteToWavFile(SampleBuffer, Settings.SaveName, const_cast<FString&>(FilePath),
       []() {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("Audio Buffer Saved to WAV"));
    });
}
```

4.50.3.7 StartCapturingAudio()

```
void UAudioManager::StartCapturingAudio ( )
```

Starts The Capturing of Audio onto the Buffer.

Definition at line 39 of file AudioManager.cpp.

```
{
    AudioBuffer.Empty();

    bIsCapturingAudio = true;
}
```

4.50.3.8 StopCapturingAudio()

```
void UAudioManager::StopCapturingAudio ( )
```

Stops the Capturing of Audio onto the Buffer, and sends the audio to the transcription service.

Definition at line 46 of file AudioManager.cpp.

```
{
    bIsCapturingAudio = false;

    if (AudioBuffer.Num() == 0)
        return;

    SaveAudioBufferToWAV(Settings.SavePath);

    if (OnAudioReadyForTranscription.ExecuteIfBound(AudioBuffer))
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Executing Audio Ready For Transcription
    Delegate. ||"));
    }
    else
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| No Delegates Bound to Audio Ready For
    Transcription Delegate. ||"));
    }

    AudioBuffer.Empty();
}
```

4.50.4 Member Data Documentation

4.50.4.1 OnAudioReadyForTranscription

`TDelegate<void(const TArray<float>)> UAudioManager::OnAudioReadyForTranscription`

Delegate for when the AudioBuffer is Ready To Be Sent For Transcription.

Definition at line 124 of file AudioManager.h.

4.50.4.2 Settings

`FAudioManagerSettings UAudioManager::Settings`

The Settings of the Audio Manager.

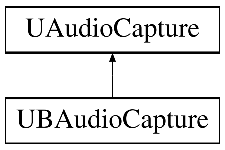
Definition at line 119 of file AudioManager.h.

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

- Source/OpenAccessibilityCommunication/Public/AudioManager.h
- Source/OpenAccessibilityCommunication/Private/AudioManager.cpp

4.51 UBAudioCapture Class Reference

Inheritance diagram for UBAudioCapture:



Public Member Functions

- bool OpenDefaultAudioStream (int32 OverrideSampleRate, int32 OverrideInputChannels)

  Opens the default audio stream.

4.51.1 Detailed Description

Definition at line 11 of file UBAudioCapture.h.

4.51.2 Constructor & Destructor Documentation

4.51.2.1 UBAudioCapture()

```
UBAudioCapture::UBAudioCapture ( )
```

Definition at line 6 of file UBAudioCapture.cpp.

```
00006                                   : UAudioCapture()
00007 {
00008
00009 }
```

4.51.2.2 ∼UBAudioCapture()

```
UBAudioCapture::∼UBAudioCapture ( )
```

Definition at line 11 of file UBAudioCapture.cpp.

```
00012 {
00013 }
```

4.51.3 Member Function Documentation

4.51.3.1 OpenDefaultAudioStream()

```
bool UBAudioCapture::OpenDefaultAudioStream (
            int32 OverrideSampleRate,
            int32 OverrideInputChannels )
```

Opens the default audio stream.

Parameters

OverrideSampleRate	Override for the Audiobuffers Sample Rate.
OverrideInputChannels	Override for the Amount of Input Channel Amount.

Returns

True, if the Audiostream was opened correctly. False, if the Audio Stream could not be opened.

Definition at line 15 of file UBAudioCapture.cpp.

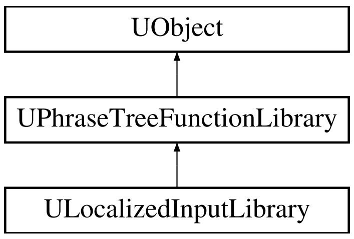
```
{
    if (!AudioCapture.IsStreamOpen())
    {
        if (!AudioCapture.IsStreamOpen())
        {
            Audio::FOnAudioCaptureFunction OnCapture = [this](const void* AudioData, int32 NumFrames, int32 InNumChannels, int32 InSampleRate, double StreamTime, bool bOverFlow)
                {
                    OnGeneratedAudio((const float*)AudioData, NumFrames * InNumChannels);
                };

            // Start the stream here to avoid hitching the audio render thread.
            Audio::FAudioCaptureDeviceParams Params;
            if (OverrideSampleRate != NULL)
                Params.SampleRate = OverrideSampleRate;
            if (OverrideInputChannels != NULL)
                Params.NumInputChannels = OverrideInputChannels;


            if (AudioCapture.OpenAudioCaptureStream(Params, MoveTemp(OnCapture), 1024))
            {
                // If we opened the capture stream succesfully, get the capture device info and initialize the UAudioGenerator
                Audio::FCaptureDeviceInfo Info;
                if (AudioCapture.GetCaptureDeviceInfo(Info))
                {
                    Init(
                        OverrideSampleRate != NULL ? OverrideSampleRate : Info.PreferredSampleRate ,
                        OverrideInputChannels != NULL ? OverrideInputChannels : Info.InputChannels
                    );

                    return true;
                }
            }
        }

        return false;
    }

    return false;
}
```

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

- Source/OpenAccessibilityCommunication/Public/UBAudioCapture.h
- Source/OpenAccessibilityCommunication/Private/UBAudioCapture.cpp

4.52 ULocalizedInputLibrary Class Reference

Inheritance diagram for ULocalizedInputLibrary:



Public Member Functions

- ULocalizedInputLibrary (const FObjectInitializer &ObjectInitializer)
- virtual void BindBranches (TSharedRef< FPhraseTree > PhraseTree) override
- void KeyboardInputAdd (FParseRecord &Record)
- void KeyboardInputRemove (FParseRecord &Record)
- void KeyboardInputReset (FParseRecord &Record)
- void KeyboardInputConfirm (FParseRecord &Record)
- void KeyboardInputExit (FParseRecord &Record)

4.52.1 Detailed Description

Definition at line 12 of file LocalizedInputLibrary.h.

4.52.2 Constructor & Destructor Documentation

4.52.2.1 ULocalizedInputLibrary()

```
ULocalizedInputLibrary::ULocalizedInputLibrary (
            const FObjectInitializer & ObjectInitializer )
```

Definition at line 13 of file LocalizedInputLibrary.cpp.

```
00014 {
00015
00016 }
```

4.52.2.2 ∼ULocalizedInputLibrary()

```
ULocalizedInputLibrary::∼ULocalizedInputLibrary ( )  [virtual]
```

Definition at line 18 of file LocalizedInputLibrary.cpp.

```
00019 {
00020
00021 }
```

4.52.3 Member Function Documentation

4.52.3.1 BindBranches()

```
void ULocalizedInputLibrary::BindBranches (
            TSharedRef< FPhraseTree > PhraseTree )  [override], [virtual]
```

Binds Branches originating from this Library onto the provided Phrase Tree.

Parameters

PhraseTree	Reference to the PhraseTree to Bind this Library to.

Reimplemented from UPhraseTreeFunctionLibrary.

Definition at line 23 of file LocalizedInputLibrary.cpp.

```
{
    PhraseTree->BindBranch(
        MakeShared<FPhraseNode>(TEXT("INPUT"),
        TPhraseNodeArray {

            MakeShared<FPhraseNode>(TEXT("ADD"),
            TPhraseNodeArray {

                MakeShared<FPhraseStringInputNode>(TEXT("PHRASE_TO_ADD"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&ULocalizedInputLibrary::KeyboardInputAdd))

                })

            }),

            MakeShared<FPhraseNode>(TEXT("REMOVE"),
            TPhraseNodeArray {

                MakeShared<FPhraseInputNode<int32>>(TEXT("AMOUNT"),
                TPhraseNodeArray {


MakeShared<FPhraseEventNode>(CreateParseDelegate(this,&ULocalizedInputLibrary::KeyboardInputRemove))

                })

            }),

            MakeShared<FPhraseNode>(TEXT("RESET"),
            TPhraseNodeArray {

                MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&ULocalizedInputLibrary::KeyboardInputReset))

            }),

            /*
            MakeShared<FPhraseNode>(TEXT("CONFIRM"),
            TPhraseNodeArray {

                MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&ULocalizedInputLibrary::KeyboardInputConfirm))

            }),
            */

            MakeShared<FPhraseNode>(TEXT("EXIT"),
            TPhraseNodeArray {

                MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&ULocalizedInputLibrary::KeyboardInputExit))

            })

        })
    );
}
```

4.52.3.2 KeyboardInputAdd()

```
void ULocalizedInputLibrary::KeyboardInputAdd (
            FParseRecord & Record )
```

Phrase Event for Adding String Words to the Active Keyboard Focus.

Parameters

Record	The ParseRecord accumulated until this Event.

Definition at line 80 of file LocalizedInputLibrary.cpp.

```
                                                                    {
    GET_ACTIVE_KEYBOARD_WIDGET(KeyboardFocusedWidget);

    FString WidgetType = KeyboardFocusedWidget->GetTypeAsString();

    UParseStringInput *PhraseInput = Record.GetPhraseInput<UParseStringInput>(TEXT("PHRASE_TO_ADD"));
    if (PhraseInput == nullptr)
        return;

    if (WidgetType == "SEditableText")
    {
        TSharedPtr<SEditableText> EditableText =
StaticCastSharedPtr<SEditableText>(KeyboardFocusedWidget);
        if (!EditableText.IsValid()) {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputAdd: CURRENT ACTIVE
WIDGET IS NOT OF TYPE - SEditableText"));
            return;
        }

        FString CurrText = EditableText->GetText().ToString();
        EditableText->SetText(
            FText::FromString(CurrText.TrimStartAndEnd() + TEXT(" ") + PhraseInput->GetValue())
        );
    }
    else if (WidgetType == "SMultiLineEditableText")
    {
        TSharedPtr<SMultiLineEditableText> MultiLineEditableText =
StaticCastSharedPtr<SMultiLineEditableText>(KeyboardFocusedWidget);
        if (!MultiLineEditableText.IsValid()) {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputAdd: CURRENT ACTIVE
WIDGET IS NOT OF TYPE - SMultiLineEditableText"));
            return;
        }

        FString CurrText = MultiLineEditableText->GetText().ToString();
        MultiLineEditableText->SetText(
            FText::FromString(CurrText.TrimStartAndEnd() + TEXT(" ") + PhraseInput->GetValue())
        );
    }
    else UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputAdd: CURRENT ACTIVE
WIDGET IS NOT AN INTERFACEABLE TYPE"));
}
```

4.52.3.3 KeyboardInputConfirm()

```
void ULocalizedInputLibrary::KeyboardInputConfirm (
            FParseRecord & Record )
```

Phrase Event for Submitting the Keyboard Input on the Active Keyboard Focus.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 192 of file LocalizedInputLibrary.cpp.

```
{
    GET_ACTIVE_KEYBOARD_WIDGET(KeyboardFocusedWidget);

    FName WidgetType = KeyboardFocusedWidget->GetType();

    if (WidgetType == SEditableText::StaticWidgetClass().GetWidgetType())
    {
        TSharedPtr<SEditableText> EditableText =
StaticCastSharedPtr<SEditableText>(KeyboardFocusedWidget);
```

```
        if (!EditableText.IsValid())
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputConfirm: CURRENT ACTIVE WIDGET IS NOT OF TYPE - SEditableText"))
            return;
        }

    }
    else if (WidgetType == SMultiLineEditableText::StaticWidgetClass().GetWidgetType())
    {
        TSharedPtr<SMultiLineEditableText> MultiLineEditableText = StaticCastSharedPtr<SMultiLineEditableText>(KeyboardFocusedWidget);
        if (!MultiLineEditableText.IsValid())
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputConfirm: CURRENT ACTIVE WIDGET IS NOT OF TYPE - SMultiLineEditableText"))
            return;
        }

    }
    else UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputConfirm: CURRENT ACTIVE WIDGET IS NOT AN INTERFACEABLE TYPE"))
}
```

4.52.3.4 KeyboardInputExit()

```
void ULocalizedInputLibrary::KeyboardInputExit (
            FParseRecord & Record )
```

Phrase Event for Exiting the Active Keyboard Focus, with no changes.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 221 of file LocalizedInputLibrary.cpp.

```
{
    FSlateApplication& SlateApp = FSlateApplication::Get();
    if (!SlateApp.IsInitialized())
        return;

    SlateApp.ClearKeyboardFocus();
}
```

4.52.3.5 KeyboardInputRemove()

```
void ULocalizedInputLibrary::KeyboardInputRemove (
            FParseRecord & Record )
```

Phrase Event for Removing String Chunks from the Active Keyboard Focus.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 118 of file LocalizedInputLibrary.cpp.

```
{
    GET_ACTIVE_KEYBOARD_WIDGET(KeyboardFocusedWidget);

    FString WidgetType = KeyboardFocusedWidget->GetTypeAsString();

    UParseIntInput* RemoveInput = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (RemoveInput == nullptr)
        return;

    if (WidgetType == "SEditableText")
    {
        TSharedPtr<SEditableText> EditableText =
StaticCastSharedPtr<SEditableText>(KeyboardFocusedWidget);
        if (!EditableText.IsValid()) {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputRemove: CURRENT ACTIVE
WIDGET IS NOT OF TYPE - SEditableText"));
            return;
        }

        EditableText->SetText(
            FText::FromString(
                EventUtils::RemoveWordsFromEnd(EditableText->GetText().ToString(),
RemoveInput->GetValue())
            )
        );
    }
    else if (WidgetType == "SMultiLineEditableText")
    {
        TSharedPtr<SMultiLineEditableText> MultiLineEditableText =
StaticCastSharedPtr<SMultiLineEditableText>(KeyboardFocusedWidget);
        if (!MultiLineEditableText.IsValid()) {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputRemove: CURRENT ACTIVE
WIDGET IS NOT OF TYPE - SMultiLineEditableText"));
            return;
        }

        MultiLineEditableText->SetText(
            FText::FromString(
                EventUtils::RemoveWordsFromEnd(MultiLineEditableText->GetText().ToString(),
RemoveInput->GetValue())
            )
        );
    }
    else UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputRemove: CURRENT ACTIVE
WIDGET IS NOT AN INTERFACEABLE TYPE"));
}
```

4.52.3.6 KeyboardInputReset()

```
void ULocalizedInputLibrary::KeyboardInputReset (
            FParseRecord & Record )
```

Phrase Event for Resetting the Active Keyboard Focus.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 159 of file LocalizedInputLibrary.cpp.

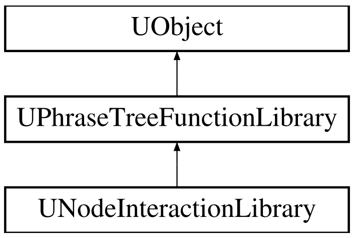
```
{
    GET_ACTIVE_KEYBOARD_WIDGET(KeyboardFocusedWidget);

    FString WidgetType = KeyboardFocusedWidget->GetTypeAsString();

    if (WidgetType == "SEditableText")
    {
        TSharedPtr<SEditableText> EditableText =
StaticCastSharedPtr<SEditableText>(KeyboardFocusedWidget);
        if (!EditableText.IsValid()) {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputReset: CURRENT ACTIVE
WIDGET IS NOT OF TYPE - SEditableText"));
            return;
        }

        EditableText->SetText(
            FText::FromString(TEXT(""))
        );
    }
    else if (WidgetType == "SMultiLineEditableText")
    {
        TSharedPtr<SMultiLineEditableText> MultiLineEditableText =
StaticCastSharedPtr<SMultiLineEditableText>(KeyboardFocusedWidget);
        if (!MultiLineEditableText.IsValid()) {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputReset: CURRENT ACTIVE
WIDGET IS NOT OF TYPE - SMultiLineEditableText"));
            return;
        }

        MultiLineEditableText->SetText(
            FText::FromString(TEXT(""))
        );
    }
    else UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputReset: CURRENT ACTIVE
WIDGET IS NOT AN INTERFACEABLE TYPE"));
}
```

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

- Source/OpenAccessibility/Public/PhraseEvents/LocalizedInputLibrary.h
- Source/OpenAccessibility/Private/PhraseEvents/LocalizedInputLibrary.cpp

4.53 UNodeInteractionLibrary Class Reference

Inheritance diagram for UNodeInteractionLibrary:



Public Member Functions

- UNodeInteractionLibrary (const FObjectInitializer &ObjectInitializer)
- virtual void BindBranches (TSharedRef< FPhraseTree > PhraseTree) override
- void MoveNode (FParseRecord &Record)
- void DeleteNode (FParseRecord &Record)
- void NodeIndexFocus (int32 Index)
- void PinConnect (FParseRecord &Record)
- void PinDisconnect (FParseRecord &Record)
- TSharedPtr< IMenu > NodeAddMenu (FParseRecord &Record)
- TSharedPtr< IMenu > NodeAddPinMenu (FParseRecord &Record)
- void NodeAddSelect (FParseRecord &Record)
- void NodeAddSearchAdd (FParseRecord &Record)
- void NodeAddSearchRemove (FParseRecord &Record)
- void NodeAddSearchReset (FParseRecord &Record)
- void NodeAddScroll (FParseRecord &Record)
- void SelectionNodeToggle (FParseRecord &Record)
- void SelectionReset (FParseRecord &Record)
- void SelectionMove (FParseRecord &Record)
- void SelectionAlignment (FParseRecord &Record)

- void SelectionStraighten (FParseRecord &Record)
- void SelectionComment (FParseRecord &Record)
- void LocomotionSelect (FParseRecord &Record)
- void LocomotionRevert (FParseRecord &Record)
- void LocomotionConfirm (FParseRecord &Record)
- void LocomotionCancel (FParseRecord &Record)
- void BlueprintCompile (FParseRecord &Record)

4.53.1 Detailed Description

Definition at line 12 of file NodeInteractionLibrary.h.

4.53.2 Constructor & Destructor Documentation

4.53.2.1 UNodeInteractionLibrary()

```
UNodeInteractionLibrary::UNodeInteractionLibrary (
            const FObjectInitializer & ObjectInitializer )
```

Definition at line 21 of file NodeInteractionLibrary.cpp.

```
00022     : Super(ObjectInitializer)
00023 {
00024
00025 }
```

4.53.2.2 ∼UNodeInteractionLibrary()

```
UNodeInteractionLibrary::∼UNodeInteractionLibrary ( )  [virtual]
```

Definition at line 27 of file NodeInteractionLibrary.cpp.

```
00028 {
00029
00030 }
```

4.53.3 Member Function Documentation

4.53.3.1 BindBranches()

```
void UNodeInteractionLibrary::BindBranches (
            TSharedRef< FPhraseTree > PhraseTree )  [override], [virtual]
```

Binds Branches originating from this Library onto the provided Phrase Tree.

Parameters

PhraseTree	Reference to the PhraseTree to Bind this Library to.

Reimplemented from UPhraseTreeFunctionLibrary.

Definition at line 32 of file NodeInteractionLibrary.cpp.

```
{
    // Events
    TDelegate<void(int32)> NodeIndexFocusDelegate = CreateInputDelegate(this,
  &UNodeInteractionLibrary::NodeIndexFocus);


    // Add Node Children Branch
    TPhraseNodeArray AddNodeContextChildren = TPhraseNodeArray {

        MakeShared<FPhraseNode>(TEXT("SELECT"),
        TPhraseNodeArray {

            MakeShared<FPhraseInputNode<int32»(TEXT("SELECTION_INDEX"),
            TPhraseNodeArray {

                MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
  &UNodeInteractionLibrary::NodeAddSelect))

            })

        }),

        MakeShared<FPhraseNode>(TEXT("SEARCH"),
        TPhraseNodeArray{

            MakeShared<FPhraseNode>(TEXT("ADD"),
            TPhraseNodeArray {

                MakeShared<FPhraseStringInputNode>(TEXT("SEARCH_PHRASE"),
                TPhraseNodeArray{

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
  &UNodeInteractionLibrary::NodeAddSearchAdd))

                })

            }),

            MakeShared<FPhraseNode>(TEXT("REMOVE"),
            TPhraseNodeArray {

                MakeShared<FPhraseInputNode<int32»(TEXT("AMOUNT"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
  &UNodeInteractionLibrary::NodeAddSearchRemove))

                })

            }),

            MakeShared<FPhraseNode>(TEXT("RESET"),
            TPhraseNodeArray {

                MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
  &UNodeInteractionLibrary::NodeAddSearchReset))

            })

        }),

        MakeShared<FPhraseNode>(TEXT("SCROLL"),
        TPhraseNodeArray {

            MakeShared<FPhraseScrollInputNode>(TEXT("DIRECTION"),
            TPhraseNodeArray {

                MakeShared<FPhraseInputNode<int32»(TEXT("AMOUNT"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
  &UNodeInteractionLibrary::NodeAddScroll))

                })

            }),

        }),

    };

    PhraseTree->BindBranches(
        TPhraseNodeArray
        {
            MakeShared<FPhraseNode>(TEXT("NODE"),
            TPhraseNodeArray {

                MakeShared<FPhraseInputNode<int32»(TEXT("NODE_INDEX"),
                TPhraseNodeArray {

                    MakeShared<FPhraseNode>(TEXT("MOVE"),
                    TPhraseNodeArray {

                        MakeShared<FPhrase2DDirectionalInputNode>(TEXT("DIRECTION"),
                        TPhraseNodeArray {

                            MakeShared<FPhraseInputNode<int32»(TEXT("AMOUNT"),
                            TPhraseNodeArray {

                                MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::MoveNode))

                            })

                        })

                    }),

                    MakeShared<FPhraseNode>(TEXT("REMOVE"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::DeleteNode))

                    }),

                    MakeShared<FPhraseInputNode<int32>>(TEXT("PIN_INDEX"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseNode>(TEXT("CONNECT"),
                        TPhraseNodeArray {

                            MakeShared<FPhraseContextMenuNode<UAccessibilityGraphEditorContext»(
                                TEXT("ADD"),
                                1.5f,
                                CreateMenuDelegate(this, &UNodeInteractionLibrary::NodeAddPinMenu),
                                AddNodeContextChildren
                            ),

                            MakeShared<FPhraseInputNode<int32>>(TEXT("NODE_INDEX"),
                            TPhraseNodeArray {

                                MakeShared<FPhraseInputNode<int32»(TEXT("PIN_INDEX"),
                                TPhraseNodeArray {

                                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::PinConnect))

                                })

                            }, NodeIndexFocusDelegate)

                        }),

                        MakeShared<FPhraseNode>(TEXT("DISCONNECT"),
                        TPhraseNodeArray {

                            MakeShared<FPhraseInputNode<int32»(TEXT("NODE_INDEX"),
                            TPhraseNodeArray {

                                MakeShared<FPhraseInputNode<int32»(TEXT("PIN_INDEX"),
                                TPhraseNodeArray {

                                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::PinDisconnect))

                                })

                            })

                        })

                    })

                }, NodeIndexFocusDelegate),

                MakeShared<FPhraseNode>(TEXT("SELECT"),
                TPhraseNodeArray {

                    MakeShared<FPhraseInputNode<int32>>(TEXT("NODE_INDEX"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::SelectionNodeToggle))

                    }),

                    MakeShared<FPhraseNode>(TEXT("RESET"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::SelectionReset))

                    }),

                    MakeShared<FPhraseNode>(TEXT("MOVE"),
                    TPhraseNodeArray {

                        MakeShared<FPhrase2DDirectionalInputNode>(TEXT("DIRECTION"),
                        TPhraseNodeArray {

                            MakeShared<FPhraseInputNode<int32>>(TEXT("AMOUNT"),
                            TPhraseNodeArray {

                                MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::SelectionMove))

                            })

                        })

                    }),

                    MakeShared<FPhraseNode>(TEXT("STRAIGHTEN"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::SelectionStraighten))

                    }),

                    MakeShared<FPhraseNode>(TEXT("ALIGNMENT"),
                    TPhraseNodeArray {

                        MakeShared<FPhrasePositionalInputNode>(TEXT("POSITION"),
                        TPhraseNodeArray {

                            MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::SelectionAlignment))

                        })

                    }),

                    MakeShared<FPhraseNode>(TEXT("COMMENT"),
                    TPhraseNodeArray{

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::SelectionComment))

                    })

                }),

                MakeShared<FPhraseContextMenuNode<UAccessibilityGraphEditorContext>>(
                    TEXT("ADD"),
                    1.5f,
                    CreateMenuDelegate(this, &UNodeInteractionLibrary::NodeAddMenu),
                    AddNodeContextChildren
                ),

            }),

            MakeShared<FPhraseNode>(TEXT("GRAPH"),
            TPhraseNodeArray {

                MakeShared<FPhraseNode>(TEXT("COMPILE"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::BlueprintCompile))

                })),

                MakeShared<FPhraseContextNode<UAccessibilityGraphLocomotionContext>>(TEXT("MOVE"),
                TPhraseNodeArray {

                    MakeShared<FPhraseNode>(TEXT("SELECT"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseInputNode<int32»(TEXT("INDEX"),
                        TPhraseNodeArray {

                            MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::LocomotionSelect))

                        })

                    }),

                    MakeShared<FPhraseNode>(TEXT("REVERT"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::LocomotionRevert))

                    }),

                    MakeShared<FPhraseNode>(TEXT("CONFIRM"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::LocomotionConfirm))

                    }),

                    MakeShared<FPhraseNode>(TEXT("CANCEL"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::LocomotionCancel))

                    })

                }),
            })
        }
    );

};
```

4.53.3.2 BlueprintCompile()

```
void UNodeInteractionLibrary::BlueprintCompile (
            FParseRecord & Record )
```

Phrase Event for Compiling Blueprint Linked to the Active Blueprint Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 880 of file NodeInteractionLibrary.cpp.

```
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UEdGraph* ActiveGraph = ActiveGraphEditor->GetCurrentGraph();
    if (ActiveGraph == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("BlueprintCompile: Active Graph Not Found"));
        return;
    }

    UBlueprint* FoundBlueprint = FBlueprintEditorUtils::FindBlueprintForGraph(ActiveGraph);
    if (FoundBlueprint == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("BlueprintCompile: Blueprint Not Found"));
        return;
    }

    TSharedPtr<FBlueprintEditor> BlueprintEditor = StaticCastSharedPtr<FBlueprintEditor>(FKismetEditorUtilities::GetIBlueprintEditorForObject(FoundBlueprint, false));
    if (!BlueprintEditor.IsValid())
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("BlueprintCompile: BlueprintEditor Not Found"));
        return;
    }

    BlueprintEditor->Compile();
}
```

4.53.3.3 DeleteNode()

```
void UNodeInteractionLibrary::DeleteNode (
            FParseRecord & Record )
```

Phrase Event for Deleting a Node, on the Active Graph Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 395 of file NodeInteractionLibrary.cpp.

```
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UParseIntInput* IndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("NODE_INDEX"));
    if (IndexInput == nullptr)
        return;

    TSharedRef<FAssetAccessibilityRegistry> AssetRegistry = GetAssetRegistry();
    TSharedRef<FGraphIndexer> Indexer = AssetRegistry->GetGraphIndexer(ActiveGraphEditor->GetCurrentGraph());

    UEdGraphNode* Node = Indexer->GetNode(IndexInput->GetValue());
    if (Node == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("DeleteNode: Node Not Found"));
        return;
    }

    Node->Modify();
    Node->DestroyNode();
}
```

4.53.3.4 LocomotionCancel()

```
void UNodeInteractionLibrary::LocomotionCancel (
            FParseRecord & Record )
```

Phrase Event for Canceling the Active Graph Editors Locomotion Mode, reverting to viewport state before.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 873 of file NodeInteractionLibrary.cpp.

```
{
    GET_TOP_CONTEXT(Record, LocomotionContext, UAccessibilityGraphLocomotionContext);

    LocomotionContext->CancelLocomotion();
}
```

4.53.3.5 LocomotionConfirm()

```
void UNodeInteractionLibrary::LocomotionConfirm (
            FParseRecord & Record )
```

Phrase Event for Confirming the Current Viewport, on the Active Graph Editors Locomotion Mode.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 866 of file NodeInteractionLibrary.cpp.

```
{
    GET_TOP_CONTEXT(Record, LocomotionContext, UAccessibilityGraphLocomotionContext);

    LocomotionContext->ConfirmSelection();
}
```

4.53.3.6 LocomotionRevert()

```
void UNodeInteractionLibrary::LocomotionRevert (
            FParseRecord & Record )
```

Phrase Event for Reverting the Viewport to the Previous Rect, on the Active Graph Editors Locomotion Mode.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 856 of file NodeInteractionLibrary.cpp.

```
{
    GET_TOP_CONTEXT(Record, LocomotionContext, UAccessibilityGraphLocomotionContext);

    if (!LocomotionContext->RevertToPreviousView())
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("Locomotion Revert: Failed to Revert to
    Previous View."));
    }
}
```

4.53.3.7 LocomotionSelect()

```
void UNodeInteractionLibrary::LocomotionSelect (
            FParseRecord & Record )
```

Phrase Event for Selecting a Viewport Rect for Movement, on the Active Graph Editors Locomotion Mode.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 842 of file NodeInteractionLibrary.cpp.

```
{
    GET_TOP_CONTEXT(Record, LocomotionContext, UAccessibilityGraphLocomotionContext);

    UParseIntInput* ViewSelectionInput = Record.GetPhraseInput<UParseIntInput>(TEXT("INDEX"));
    if (ViewSelectionInput == nullptr)
        return;

    if (!LocomotionContext->SelectChunk(ViewSelectionInput->GetValue()))
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("Locomotion Select: Failed to Choose New
View."));
    }
}
```

4.53.3.8 MoveNode()

```
void UNodeInteractionLibrary::MoveNode (
            FParseRecord & Record )
```

Phrase Event for Moving a Node, on the Active Graph Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 314 of file NodeInteractionLibrary.cpp.

```
                                                                {
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UParseIntInput* IndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("NODE_INDEX"));
    UParseEnumInput* DirectionInput = Record.GetPhraseInput<UParseEnumInput>(TEXT("DIRECTION"));
    UParseIntInput* AmountInput = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (IndexInput == nullptr || DirectionInput == nullptr || AmountInput == nullptr)
        return;

    TSharedRef<FAssetAccessibilityRegistry> AssetRegistry = GetAssetRegistry();
    TSharedRef<FGraphIndexer> Indexer =
AssetRegistry->GetGraphIndexer(ActiveGraphEditor->GetCurrentGraph());

    UEdGraphNode* Node = Indexer->GetNode(IndexInput->GetValue());
    if (Node == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("MoveNode: Node Not Found"));
        return;
    }

    FVector2D PositionDelta = FVector2D::ZeroVector;
    switch (EPhrase2DDirectionalInput(DirectionInput->GetValue()))
    {
        case EPhrase2DDirectionalInput::UP:
            PositionDelta.Y -= AmountInput->GetValue();
            break;

        case EPhrase2DDirectionalInput::DOWN:
            PositionDelta.Y += AmountInput->GetValue();
            break;

        case EPhrase2DDirectionalInput::LEFT:
            PositionDelta.X -= AmountInput->GetValue();
            break;

        case EPhrase2DDirectionalInput::RIGHT:
            PositionDelta.X += AmountInput->GetValue();
            break;

        default:
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("MoveNode: Invalid Direction"));
            return;
    }

    SGraphPanel* GraphPanel = ActiveGraphEditor->GetGraphPanel();
    if (GraphPanel == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("MoveNode: Linked Graph Panel Not Found"));
    }

    TSharedPtr<SGraphNode> NodeWidget = GraphPanel ? GraphPanel->GetNodeWidgetFromGuid(Node->NodeGuid) : TSharedPtr<SGraphNode>();
    if (NodeWidget.IsValid())
    {
        SNodePanel::SNode::FNodeSet NodeFilter;
        NodeWidget->MoveTo(FVector2D(Node->NodePosX, Node->NodePosY) + PositionDelta, NodeFilter);
    }
    else
    {
        Node->Modify();
        Node->NodePosX += PositionDelta.X;
        Node->NodePosY += PositionDelta.Y;
    }

    // Move Comment Node Children
    // Note: This is a workaround for the MoveTo Function not calling the override in UEdGraphNode_Comment
    if (UEdGraphNode_Comment* CommentNode = Cast<UEdGraphNode_Comment>(Node))
    {
        for (UObject* _CommentChildNode : CommentNode->GetNodesUnderComment())
        {
            if (UEdGraphNode* CommentChildNode = Cast<UEdGraphNode>(_CommentChildNode))
            {
                if (!GraphPanel->SelectionManager.IsNodeSelected(CommentChildNode))
                {
                    CommentChildNode->Modify();
                    CommentChildNode->NodePosX += PositionDelta.X;
                    CommentChildNode->NodePosY += PositionDelta.Y;
                }
            }
        }
    }
}
```

4.53.3.9 NodeAddMenu()

```
TSharedPtr< IMenu > UNodeInteractionLibrary::NodeAddMenu (
            FParseRecord & Record )
```

Menu Event for Initializing the Node Add Context Menu, on the Active Graph Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Returns

A Shared Pointer to the Initialized Menu, otherwise an Invalid Shared Pointer.

Definition at line 510 of file NodeInteractionLibrary.cpp.

```
{
    GET_CAST_ACTIVE_TAB_RETURN(ActiveGraphEditor, SGraphEditor, TSharedPtr<IMenu>())

    SGraphPanel* GraphPanel = ActiveGraphEditor->GetGraphPanel();

    FVector2D SpawnLocation;
    {
        TSharedPtr<SWindow> TopLevelWindow =
FSlateApplication::Get().GetActiveTopLevelRegularWindow();

        if (TopLevelWindow.IsValid())
        {
            SpawnLocation = TopLevelWindow->GetPositionInScreen();
            FVector2D WindowSize = TopLevelWindow->GetSizeInScreen();

            SpawnLocation.X += WindowSize.X / 5;
            SpawnLocation.Y += WindowSize.Y / 5;
        }
        else
        {
            FDisplayMetrics DisplayMetrics;
            FSlateApplication::Get().GetDisplayMetrics(DisplayMetrics);

            SpawnLocation = FVector2D(
                DisplayMetrics.PrimaryDisplayWidth / 5,
                DisplayMetrics.PrimaryDisplayHeight / 5
            );
        }

        TSharedPtr<SWidget> ContextWidgetToFocus = GraphPanel->SummonContextMenu(
            SpawnLocation,
            GraphPanel->GetPastePosition(),
            nullptr,
            nullptr,
            TArray<UEdGraphPin *>()
        );

        if (!ContextWidgetToFocus.IsValid())
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeAddMenu: Context Keyboard Focus
Widget Not Found"));
            return TSharedPtr<IMenu>();
        }

        FWidgetPath KeyboardFocusPath;
        if (FSlateApplication::Get().FindPathToWidget(ContextWidgetToFocus.ToSharedRef(),
KeyboardFocusPath))
        {
            return FSlateApplication::Get().FindMenuInWidgetPath(KeyboardFocusPath);
        }
        else
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeAddMenu: IMenu Could Not Be
Found In Widget Path"))
            return TSharedPtr<IMenu>();
        }
    }
}
```

4.53.3.10 NodeAddPinMenu()

```
TSharedPtr< IMenu > UNodeInteractionLibrary::NodeAddPinMenu (
            FParseRecord & Record )
```

Menu Event for Initializing the Node Add Context Menu from a Pin Connection, on the Active Graph Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Returns

A Shared Pointer to the Initialized Menu, otherwise an Invalid Shared Pointer.

Definition at line 566 of file NodeInteractionLibrary.cpp.

```
{
    GET_CAST_ACTIVE_TAB_RETURN(ActiveGraphEditor, SGraphEditor, TSharedPtr<IMenu>())

    SGraphPanel* GraphPanel = ActiveGraphEditor->GetGraphPanel();

    FVector2D SpawnLocation;
    {
        TSharedPtr<SWindow> TopLevelWindow =
FSlateApplication::Get().GetActiveTopLevelRegularWindow();

        if (TopLevelWindow.IsValid())
        {
            SpawnLocation = TopLevelWindow->GetPositionInScreen();
            FVector2D WindowSize = TopLevelWindow->GetSizeInScreen();

            SpawnLocation.X += WindowSize.X / 5;
            SpawnLocation.Y += WindowSize.Y / 5;
        }
        else
        {
            FDisplayMetrics DisplayMetrics;
            FSlateApplication::Get().GetDisplayMetrics(DisplayMetrics);

            SpawnLocation = FVector2D(
                DisplayMetrics.PrimaryDisplayWidth / 5,
                DisplayMetrics.PrimaryDisplayHeight / 5
            );
        }

        TSharedRef<FGraphIndexer> Indexer =
GetAssetRegistry()->GetGraphIndexer(ActiveGraphEditor->GetCurrentGraph());

        UParseIntInput* NodeIndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("NODE_INDEX"));
        UParseIntInput* PinIndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("PIN_INDEX"));

        if (NodeIndexInput == nullptr || PinIndexInput == nullptr)
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeAddMenu: Invalid Inputs"));

            return TSharedPtr<IMenu>();
        }

        TSharedPtr<SWidget> ContextWidgetToFocus = GraphPanel->SummonContextMenu(
            SpawnLocation,
            GraphPanel->GetPastePosition(),
            nullptr,
            nullptr,
            TArray<UEdGraphPin*> {
                Indexer->GetPin(
                    NodeIndexInput->GetValue(),
                    PinIndexInput->GetValue()
                )
            }
        );

        if (!ContextWidgetToFocus.IsValid())
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeAddMenu: Context Keyboard Focus
Widget Not Found"));
            return TSharedPtr<IMenu>();
        }

        FWidgetPath KeyboardFocusPath;
        if (FSlateApplication::Get().FindPathToWidget(ContextWidgetToFocus.ToSharedRef(),
KeyboardFocusPath))
        {
            return FSlateApplication::Get().FindMenuInWidgetPath(KeyboardFocusPath);
        }
        else
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeAddMenu: IMenu Could Not Be
Found In Widget Path"))
            return TSharedPtr<IMenu>();
        }
    }
}
```

4.53.3.11 NodeAddScroll()

```
void UNodeInteractionLibrary::NodeAddScroll (
            FParseRecord & Record )
```

Phrase Event for Applying Movement to the Scrollbar of the Active Graph Editors Node Add Context Menu.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 681 of file NodeInteractionLibrary.cpp.

```
{
    GET_TOP_CONTEXT(Record, ContextMenu, UAccessibilityGraphEditorContext)

    UParseEnumInput* DirectionInput = Record.GetPhraseInput<UParseEnumInput>(TEXT("DIRECTION"));
    UParseIntInput* AmountInput = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (DirectionInput == nullptr || AmountInput == nullptr)
        return;

    switch (EPhraseScrollInput(DirectionInput->GetValue()))
    {
        case EPhraseScrollInput::UP:
            ContextMenu->AppendScrollDistance(-AmountInput->GetValue());
            break;

        case EPhraseScrollInput::DOWN:
            ContextMenu->AppendScrollDistance(AmountInput->GetValue());
            break;

        case EPhraseScrollInput::TOP:
            ContextMenu->SetScrollDistanceTop();
            break;

        case EPhraseScrollInput::BOTTOM:
            ContextMenu->SetScrollDistanceBottom();
                break;

        default:
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeAddScroll: Invalid Scroll
       Position / Direction"));
            return;
    }
}
```

4.53.3.12 NodeAddSearchAdd()

```
void UNodeInteractionLibrary::NodeAddSearchAdd (
            FParseRecord & Record )
```

Phrase Event for Appending Strings to the SearchBar on the Active Graph Editors Node Add Context Menu.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 650 of file NodeInteractionLibrary.cpp.

```
{
    GET_TOP_CONTEXT(Record, ContextMenu, UAccessibilityGraphEditorContext)

    UParseStringInput *SearchInput = Record.GetPhraseInput<UParseStringInput>(TEXT("SEARCH_PHRASE"));
    if (SearchInput == nullptr)
        return;

    ContextMenu->AppendFilterText(SearchInput->GetValue());
}
```

4.53.3.13 NodeAddSearchRemove()

```
void UNodeInteractionLibrary::NodeAddSearchRemove (
            FParseRecord & Record )
```

Phrase Event for Removing String Chunks on the SearchBar of the Active Graph Editors Node Add Context Menu.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 661 of file NodeInteractionLibrary.cpp.

```
{
    GET_TOP_CONTEXT(Record, ContextMenu, UAccessibilityGraphEditorContext);

    UParseIntInput* RemoveAmountInput = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (RemoveAmountInput == nullptr)
        return;

    ContextMenu->SetFilterText(
        EventUtils::RemoveWordsFromEnd(ContextMenu->GetFilterText(), RemoveAmountInput->GetValue())
    );
}
```

4.53.3.14 NodeAddSearchReset()

```
void UNodeInteractionLibrary::NodeAddSearchReset (
            FParseRecord & Record )
```

Phrase Event for Resetting the SearchBar of the Active Graph Editors Node Add Context Menu.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 674 of file NodeInteractionLibrary.cpp.

```
{
    GET_TOP_CONTEXT(Record, ContextMenu, UAccessibilityGraphEditorContext)

    ContextMenu->SetFilterText(TEXT(""));
}
```

4.53.3.15 NodeAddSelect()

```
void UNodeInteractionLibrary::NodeAddSelect (
            FParseRecord & Record )
```

Phrase Event for Selecting an Item on the Active Graph Editors Node Add Context Menu.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 639 of file NodeInteractionLibrary.cpp.

```
{
    GET_TOP_CONTEXT(Record, ContextMenu, UAccessibilityGraphEditorContext)

    UParseIntInput* IndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("SELECTION_INDEX"));
    if (IndexInput == nullptr)
        return;

    ContextMenu->SelectAction(IndexInput->GetValue());
}
```

4.53.3.16 NodeIndexFocus()

```
void UNodeInteractionLibrary::NodeIndexFocus (
            int32 Index )
```

Input Event for Adding the specified Node Index to the Active Selection Set.

Parameters

Index	The Index Provided Through Voice Input.

Definition at line 417 of file NodeInteractionLibrary.cpp.

```
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    TSharedRef<FGraphIndexer> Indexer = GetAssetRegistry()->GetGraphIndexer(
        ActiveGraphEditor->GetCurrentGraph()
    );

    UEdGraphNode* Node = Indexer->GetNode(Index);
    if (Node == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeSelectionFocus: Node Not Found"));
        return;
    }

    ActiveGraphEditor->SetNodeSelection(Node, true);
}
```

4.53.3.17 PinConnect()

```
void UNodeInteractionLibrary::PinConnect (
            FParseRecord & Record )
```

Phrase Event for Connecting Two Provided Pins, on the Active Graph Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 435 of file NodeInteractionLibrary.cpp.

```
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UEdGraph* Graph = ActiveGraphEditor->GetCurrentGraph();

    TArray<UParseInput*> NodeInputs = Record.GetPhraseInputs(TEXT("NODE_INDEX"));
    TArray<UParseInput*> PinInputs = Record.GetPhraseInputs(TEXT("PIN_INDEX"));

    if (NodeInputs.Num() != 2 || PinInputs.Num() != 2)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("PinConnect: Invalid Inputs Amount"));
        return;
    }

    TSharedRef<FGraphIndexer> Indexer = GetAssetRegistry()->GetGraphIndexer(Graph);

    UEdGraphPin* SourcePin = Indexer->GetPin(
        Cast<UParseIntInput>(NodeInputs[0])->GetValue(),
        Cast<UParseIntInput>(PinInputs[0])->GetValue()
    );

    UEdGraphPin* TargetPin = Indexer->GetPin(
        Cast<UParseIntInput>(NodeInputs[1])->GetValue(),
        Cast<UParseIntInput>(PinInputs[1])->GetValue()
    );

    if (SourcePin == nullptr || TargetPin == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("PinConnect: Pins Not Found"));
        return;
    }

    if (!Graph->GetSchema()->TryCreateConnection(SourcePin, TargetPin))
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("PinConnect: Pin Connection Failed"));
    }
}
```

4.53.3.18 PinDisconnect()

```
void UNodeInteractionLibrary::PinDisconnect (
              FParseRecord & Record )
```

Phrase Event for Disconnecting Two Provided Pins, on the Active Graph Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 474 of file NodeInteractionLibrary.cpp.

```
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UEdGraph* Graph = ActiveGraphEditor->GetCurrentGraph();

    TArray<UParseInput*> NodeInputs = Record.GetPhraseInputs(TEXT("NODE_INDEX"));
    TArray<UParseInput*> PinInputs = Record.GetPhraseInputs(TEXT("PIN_INDEX"));

    if (NodeInputs.Num() != 2 || PinInputs.Num() != 2)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("PinDisconnect: Invalid Inputs
       Amount"));
        return;
    }

    TSharedRef<FGraphIndexer> Indexer = GetAssetRegistry()->GetGraphIndexer(Graph);

    UEdGraphPin* SourcePin = Indexer->GetPin(
        Cast<UParseIntInput>(NodeInputs[0])->GetValue(),
        Cast<UParseIntInput>(PinInputs[0])->GetValue()
    );

    UEdGraphPin* TargetPin = Indexer->GetPin(
        Cast<UParseIntInput>(NodeInputs[1])->GetValue(),
        Cast<UParseIntInput>(PinInputs[1])->GetValue()
    );

    if (SourcePin == nullptr || TargetPin == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("PinDisconnect: Pins Not Found"));
        return;
    }

    Graph->GetSchema()->BreakSinglePinLink(SourcePin, TargetPin);
}
```

4.53.3.19 SelectionAlignment()

```
void UNodeInteractionLibrary::SelectionAlignment (
            FParseRecord & Record )
```

Phrase Event for Aligning the Selection Sets Nodes, on the Active Graph Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 785 of file NodeInteractionLibrary.cpp.

```
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UParseEnumInput* PositionInput = Record.GetPhraseInput<UParseEnumInput>(TEXT("POSITION"));
    if (PositionInput == nullptr)
        return;

    switch (EPhrasePositionalInput(PositionInput->GetValue()))
    {
        case EPhrasePositionalInput::TOP:
            ActiveGraphEditor->OnAlignTop();
            break;

        case EPhrasePositionalInput::MIDDLE:
            ActiveGraphEditor->OnAlignMiddle();
            break;

        case EPhrasePositionalInput::BOTTOM:
            ActiveGraphEditor->OnAlignBottom();
            break;

        case EPhrasePositionalInput::LEFT:
            ActiveGraphEditor->OnAlignLeft();
            break;

        case EPhrasePositionalInput::RIGHT:
            ActiveGraphEditor->OnAlignRight();
            break;

        case EPhrasePositionalInput::CENTER:
            ActiveGraphEditor->OnAlignCenter();
            break;
    }
}
```

4.53.3.20 SelectionComment()

```
void UNodeInteractionLibrary::SelectionComment (
            FParseRecord & Record )
```

Phrase Event for Applying a Comment Node Around the Selection Set, on the Active Graph Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 828 of file NodeInteractionLibrary.cpp.

```
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UEdGraph* Graph = ActiveGraphEditor->GetCurrentGraph();

    TSharedPtr<FEdGraphSchemaAction> CommentCreateAction =
Graph->GetSchema()->GetCreateCommentAction();
    if (CommentCreateAction.IsValid())
    {
        CommentCreateAction->PerformAction(Graph, nullptr, FVector2D(0, 0), true);
    }
    else UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("SelectionComment: Comment Creation
Failed"));
}
```

4.53.3.21 SelectionMove()

```
void UNodeInteractionLibrary::SelectionMove (
            FParseRecord & Record )
```

Phrase Event for Moving the Selection Set, on the Active Graph Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 745 of file NodeInteractionLibrary.cpp.

```
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UParseEnumInput* Direction = Record.GetPhraseInput<UParseEnumInput>(TEXT("DIRECTION"));
    UParseIntInput* Amount = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (Direction == nullptr || Amount == nullptr)
        return;

    for (UObject* NodeObj : ActiveGraphEditor->GetSelectedNodes())
    {
        UEdGraphNode* Node = Cast<UEdGraphNode>(NodeObj);
        if (Node == nullptr)
            continue;

        switch (EPhrase2DDirectionalInput(Direction->GetValue()))
        {
            case EPhrase2DDirectionalInput::UP:
                Node->NodePosY -= Amount->GetValue();
                break;

            case EPhrase2DDirectionalInput::DOWN:
                Node->NodePosY += Amount->GetValue();
                break;

            case EPhrase2DDirectionalInput::LEFT:
                Node->NodePosX -= Amount->GetValue();
                break;

            case EPhrase2DDirectionalInput::RIGHT:
                Node->NodePosX += Amount->GetValue();
                break;

            default:
                UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("SelectionMove: Invalid
Direction"));
                return;
        }
    }
}
```

4.53.3.22 SelectionNodeToggle()

```
void UNodeInteractionLibrary::SelectionNodeToggle (
            FParseRecord & Record )
```

Phrase Event for Toggling the specified Nodes Selection State, on the Active Graph Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 714 of file NodeInteractionLibrary.cpp.

```
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor);

    UParseIntInput* IndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("NODE_INDEX"));
    if (IndexInput == nullptr)
        return;

    TSharedRef<FGraphIndexer> Indexer = GetAssetRegistry()->GetGraphIndexer(
        ActiveGraphEditor->GetCurrentGraph()
    );

    UEdGraphNode* Node = Indexer->GetNode(IndexInput->GetValue());
    if (Node == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("SelectionToggle: Node Not Found"));
        return;
    }

    ActiveGraphEditor->SetNodeSelection(
        Node,
        !ActiveGraphEditor->GetSelectedNodes().Contains(Node)
    );
}
```

4.53.3.23 SelectionReset()

```
void UNodeInteractionLibrary::SelectionReset (
            FParseRecord & Record )
```

Phrase Event for Resetting the Selection Set, on the Active Graph Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 739 of file NodeInteractionLibrary.cpp.

```
                                                            {
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    ActiveGraphEditor->ClearSelectionSet();
}
```

4.53.3.24 SelectionStraighten()

```
void UNodeInteractionLibrary::SelectionStraighten (
            FParseRecord & Record )
```

Phrase Event for Straightening the Selection Sets Connections, on the Active Graph Editor.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 821 of file NodeInteractionLibrary.cpp.

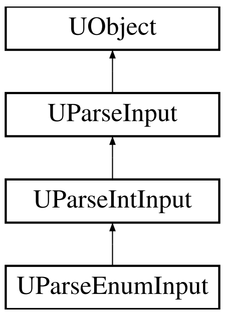
```
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    ActiveGraphEditor->OnStraightenConnections();
}
```

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

- Source/OpenAccessibility/Public/PhraseEvents/NodeInteractionLibrary.h
- Source/OpenAccessibility/Private/PhraseEvents/NodeInteractionLibrary.cpp

4.54 UParseEnumInput Class Reference

Inheritance diagram for UParseEnumInput:



Public Member Functions

- void SetEnumType (UEnum ∗InEnumType)
    Sets the Enum Type for the Input.
- void GetEnumType (UEnum ∗&OutEnumType)
    Gets the EnumType Bound To This Input.
- UEnum ∗ GetEnumType ()
    Gets the EnumType Bound To This Input.

Protected Attributes

- UEnum ∗ EnumType

4.54.1 Detailed Description

Definition at line 11 of file UParseEnumInput.h.

4.54.2 Constructor & Destructor Documentation

4.54.2.1 ∼UParseEnumInput()

```
virtual UParseEnumInput::∼UParseEnumInput ( )  [inline], [virtual]
```

Definition at line 18 of file UParseEnumInput.h.

```
    {
        delete EnumType;
    };
```

4.54.3 Member Function Documentation

4.54.3.1 GetEnumType() [1/2]

```
UEnum * UParseEnumInput::GetEnumType ( )  [inline]
```

Gets the EnumType Bound To This Input.

Returns

The Bound EnumType of the Input.

Definition at line 45 of file UParseEnumInput.h.

```
    {
        return EnumType;
    }
```

4.54.3.2 GetEnumType() [2/2]

```
void UParseEnumInput::GetEnumType (
             UEnum *& OutEnumType )  [inline]
```

Gets the EnumType Bound To This Input.

Parameters

OutEnumType	The Bound EnumType To Set.

Definition at line 36 of file UParseEnumInput.h.

```
{
    OutEnumType = EnumType;
}
```

4.54.3.3 SetEnumType()

```
void UParseEnumInput::SetEnumType (
            UEnum * InEnumType )  [inline]
```

Sets the Enum Type for the Input.

Parameters

InEnumType	The Enum Type to Set this Input To.

Definition at line 27 of file UParseEnumInput.h.

```
{
    EnumType = InEnumType;
}
```

4.54.4 Member Data Documentation

4.54.4.1 EnumType

```
UEnum* UParseEnumInput::EnumType  [protected]
```

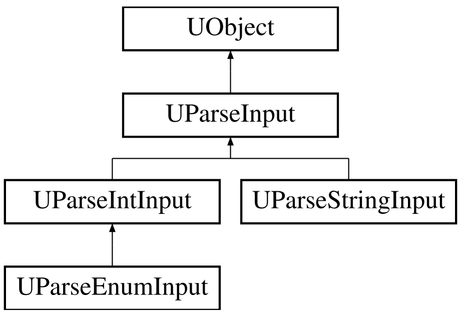
Definition at line 51 of file UParseEnumInput.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/Input/UParseEnumInput.h

4.55 UParseInput Class Reference

Inheritance diagram for UParseInput:



4.55.1 Detailed Description

Definition at line 11 of file UParseInput.h.

4.55.2 Constructor & Destructor Documentation

4.55.2.1 ∼UParseInput()

```
virtual UParseInput::∼UParseInput ( )  [inline], [virtual]
```

Definition at line 18 of file UParseInput.h.

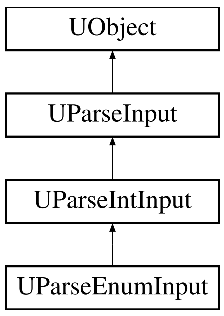
```
00019     {
00020
00021     };
```

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/Input/UParseInput.h

4.56 UParseIntInput Class Reference

Inheritance diagram for UParseIntInput:



Public Member Functions

- void SetValue (int32 InValue)

  Sets the Value of the Input.
- void GetValue (int32 &OutValue)

  Gets the Current Value of the Input.
- int32 GetValue ()

  Gets the Current Value of the Input.

Protected Attributes

- int32 Value

4.56.1 Detailed Description

Definition at line 11 of file UParseIntInput.h.

4.56.2 Constructor & Destructor Documentation

4.56.2.1 ~UParseIntInput()

```
virtual UParseIntInput::~UParseIntInput ( )  [inline], [virtual]
```

Definition at line 18 of file UParseIntInput.h.

```
    {

    };
```

4.56.3 Member Function Documentation

4.56.3.1 GetValue() [1/2]

```
int32 UParseIntInput::GetValue ( )  [inline]
```

Gets the Current Value of the Input.

Returns

The Current Value of the Input.

Definition at line 45 of file UParseIntInput.h.

```
    {
        return Value;
    }
```

4.56.3.2 GetValue() [2/2]

```
void UParseIntInput::GetValue (
            int32 & OutValue )  [inline]
```

Gets the Current Value of the Input.

Parameters

OutValue	- Returns the Current Value of the Input.

Definition at line 36 of file UParseIntInput.h.

```
00037     {
00038         OutValue = Value;
00039     }
```

4.56.3.3 SetValue()

```
void UParseIntInput::SetValue (
            int32 InValue )  [inline]
```

Sets the Value of the Input.

Parameters

InValue	- The Value to set the Input To.

Definition at line 27 of file UParseIntInput.h.

```
00028     {
00029         Value = InValue;
00030     }
```

4.56.4 Member Data Documentation

4.56.4.1 Value

```
int32 UParseIntInput::Value  [protected]
```

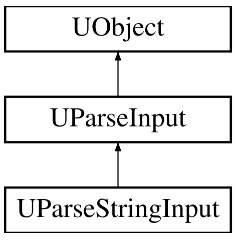
Definition at line 52 of file UParseIntInput.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/Input/UParseIntInput.h

4.57 UParseStringInput Class Reference

Inheritance diagram for UParseStringInput:



Public Member Functions

- void SetValue (FString InValue)

    Sets the Value of the Input.
- void GetValue (FString &OutValue)

    Gets the Value of the Input.
- FString GetValue ()

    Gets the Value of the Input.

Protected Attributes

- FString Value

4.57.1 Detailed Description

Definition at line 11 of file UParseStringInput.h.

4.57.2 Constructor & Destructor Documentation

4.57.2.1 ~UParseStringInput()

```
virtual UParseStringInput::~UParseStringInput ( )  [inline], [virtual]
```

Definition at line 18 of file UParseStringInput.h.

```
    {
    };
```

4.57.3 Member Function Documentation

4.57.3.1 GetValue() [1/2]

```
FString UParseStringInput::GetValue ( )  [inline]
```

Gets the Value of the Input.

Returns

Definition at line 45 of file UParseStringInput.h.

```
    {
        return Value;
    }
```

4.57.3.2 GetValue() [2/2]

```
void UParseStringInput::GetValue (
             FString & OutValue )  [inline]
```

Gets the Value of the Input.

Parameters

OutValue	- Returns the Current Value of the Input.

Definition at line 36 of file UParseStringInput.h.

```
00037     {
00038         OutValue = Value;
00039     }
```

4.57.3.3 SetValue()

```
void UParseStringInput::SetValue (
            FString InValue ) [inline]
```

Sets the Value of the Input.

Parameters

InValue	- The Value to set the Input To.

Definition at line 27 of file UParseStringInput.h.

```
00028     {
00029         Value = InValue;
00030     }
```

4.57.4 Member Data Documentation

4.57.4.1 Value

```
FString UParseStringInput::Value [protected]
```

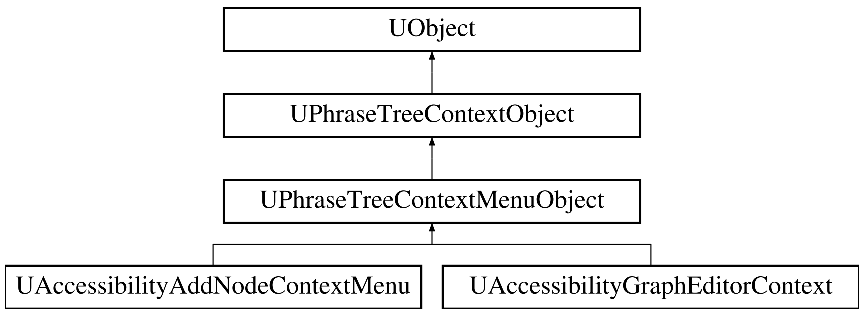
Definition at line 52 of file UParseStringInput.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/Input/UParseStringInput.h

4.58 UPhraseTreeContextMenuObject Class Reference

Inheritance diagram for UPhraseTreeContextMenuObject:


UObject
UPhraseTreeContextObject
UPhraseTreeContextMenuObject
UAccessibilityAddNodeContextMenu
UAccessibilityGraphEditorContext

Public Member Functions

- UPhraseTreeContextMenuObject (TSharedRef< IMenu > Menu)
- virtual void Init (TSharedRef< IMenu > InMenu)

  Initializes the Context Menu Object.
- virtual void Init (TSharedRef< IMenu > InMenu, TSharedRef< FPhraseNode > InContextRoot)

  Initializes the Context Menu Object.
- virtual bool Tick (float DeltaTime)
- virtual bool Close () override

  Closes the Context Menu Object.
- void BindTickDelegate ()

  Binds the Tick Delegate to the Core Ticker.
- void RemoveTickDelegate ()

  UnBinds the Tick Delegate from the Core Ticker.
- void BindMenuDismissed (TSharedRef< IMenu > InMenu)

  Binds the Menu Dismissed Callback to the Menu.
- void RemoveMenuDismissed (TSharedRef< IMenu > InMenu)

  UnBinds the Menu Dismissed Callback from the Menu.
- virtual void SetMenu (TSharedRef< IMenu > InMenu)

  Sets the Menu Object for this Context Menu.
- virtual void ScaleMenu (const float ScaleFactor)

  Scales the Provided Menu Object, and any Key Objects.

Public Attributes

- TWeakPtr< IMenu > Menu

  The Menu Object.
- TWeakPtr< SWindow > Window

  The Menu's Window.

Protected Member Functions

- TSharedPtr< SWindow > GetWindow ()

  Gets the Window Object for this Context Menu.
- void OnMenuDismissed (TSharedRef< IMenu > Menu)

  Callback for the Dismissal of the Menu.

Additional Inherited Members

4.58.1 Detailed Description

Definition at line 14 of file ContextMenuObject.h.

4.58.2 Constructor & Destructor Documentation

4.58.2.1 UPhraseTreeContextMenuObject() [1/2]

```
UPhraseTreeContextMenuObject::UPhraseTreeContextMenuObject ( )
```

Definition at line 7 of file ContextMenuObject.cpp.

```
    : UPhraseTreeContextObject()
{

}
```

4.58.2.2 UPhraseTreeContextMenuObject() [2/2]

```
UPhraseTreeContextMenuObject::UPhraseTreeContextMenuObject (
                TSharedRef< IMenu > Menu )
```

Definition at line 13 of file ContextMenuObject.cpp.

```
    : UPhraseTreeContextObject()
{

}
```

4.58.2.3 ∼UPhraseTreeContextMenuObject()

```
UPhraseTreeContextMenuObject::∼UPhraseTreeContextMenuObject ( )  [virtual]
```

Definition at line 19 of file ContextMenuObject.cpp.

```
{
    // Unbind Tick Delegate
    RemoveTickDelegate();

    if (Menu.IsValid())
        RemoveMenuDismissed(Menu.Pin().ToSharedRef());

    UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Context Menu || Destroyed ||"))
}
```

4.58.3 Member Function Documentation

4.58.3.1 BindMenuDismissed()

```
void UPhraseTreeContextMenuObject::BindMenuDismissed (
                TSharedRef< IMenu > InMenu )
```

Binds the Menu Dismissed Callback to the Menu.

Parameters

InMenu	

Definition at line 66 of file ContextMenuObject.cpp.

```
{
    MenuDismissedHandle = InMenu->GetOnMenuDismissed()
        .AddUObject(this, &UPhraseTreeContextMenuObject::OnMenuDismissed);
}
```

4.58.3.2 BindTickDelegate()

`void UPhraseTreeContextMenuObject::BindTickDelegate ( )`

Binds the Tick Delegate to the Core Ticker.

Definition at line 54 of file ContextMenuObject.cpp.

```
{
    TickDelegate = FTickerDelegate::CreateUObject(this, &UPhraseTreeContextMenuObject::Tick);
    TickDelegateHandle = FTSTicker::GetCoreTicker().AddTicker(TickDelegate);
}
```

4.58.3.3 Close()

`virtual bool UPhraseTreeContextMenuObject::Close ( ) [inline], [override], [virtual]`

Closes the Context Menu Object.

Returns

Reimplemented from UPhraseTreeContextObject.

Reimplemented in UAccessibilityAddNodeContextMenu, and UAccessibilityGraphEditorContext.

Definition at line 44 of file ContextMenuObject.h.

```
    {
        RemoveTickDelegate();
        Menu.Pin()->Dismiss();

        return true;
    };
```

4.58.3.4 GetWindow()

`TSharedPtr< SWindow > UPhraseTreeContextMenuObject::GetWindow ( ) [inline], [protected]`

Gets the Window Object for this Context Menu.

Returns

Definition at line 95 of file ContextMenuObject.h.

```
    {
        return Menu.Pin()->GetOwnedWindow();
    }
```

4.58.3.5 Init() [1/2]

```
void UPhraseTreeContextMenuObject::Init (
            TSharedRef< IMenu > InMenu ) [virtual]
```

Initializes the Context Menu Object.

Parameters

InMenu	- The Menu Object for this Context Menu.

Reimplemented in UAccessibilityAddNodeContextMenu.

Definition at line 30 of file ContextMenuObject.cpp.

```
{
    this->Menu = InMenu;
    this->Window = FSlateApplication::Get().FindWidgetWindow(
        InMenu->GetContent().ToSharedRef()
    );

    BindMenuDismissed(InMenu);
    BindTickDelegate();
}
```

4.58.3.6 Init() [2/2]

```
void UPhraseTreeContextMenuObject::Init (
              TSharedRef< IMenu > InMenu,
              TSharedRef< FPhraseNode > InContextRoot )  [virtual]
```

Initializes the Context Menu Object.

Parameters

InMenu	- The Menu Object For this Context Menu.
InContextRoot	- The Context Root In The Phrase Tree For This Object.

Reimplemented in UAccessibilityAddNodeContextMenu, and UAccessibilityGraphEditorContext.

Definition at line 41 of file ContextMenuObject.cpp.

```
{
    this->Menu = InMenu;
    this->Window = FSlateApplication::Get().FindWidgetWindow(
        InMenu->GetContent().ToSharedRef()
    );

    this->ContextRoot = InContextRoot;

    BindMenuDismissed(InMenu);
    BindTickDelegate();
}
```

4.58.3.7 OnMenuDismissed()

```
void UPhraseTreeContextMenuObject::OnMenuDismissed (
              TSharedRef< IMenu > Menu )  [protected]
```

Callback for the Dismissal of the Menu.

Parameters

Menu	

Definition at line 77 of file ContextMenuObject.cpp.

```
{
    RemoveTickDelegate();

    RemoveFromRoot();
    MarkAsGarbage();

    bIsActive = false;

    UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Context Menu || Dismissed ||"))
}
```

4.58.3.8 RemoveMenuDismissed()

```
void UPhraseTreeContextMenuObject::RemoveMenuDismissed (
            TSharedRef< IMenu > InMenu )
```

UnBinds the Menu Dismissed Callback from the Menu.

Parameters

InMenu	

Definition at line 72 of file ContextMenuObject.cpp.

```
{
    Menu.Pin()->GetOnMenuDismissed().Remove(MenuDismissedHandle);
}
```

4.58.3.9 RemoveTickDelegate()

```
void UPhraseTreeContextMenuObject::RemoveTickDelegate ( )
```

UnBinds the Tick Delegate from the Core Ticker.

Definition at line 60 of file ContextMenuObject.cpp.

```
{
    if (TickDelegateHandle != NULL)
        FTSTicker::GetCoreTicker().RemoveTicker(TickDelegateHandle);
}
```

4.58.3.10 ScaleMenu()

```
virtual void UPhraseTreeContextMenuObject::ScaleMenu (
            const float ScaleFactor )  [inline], [virtual]
```

Scales the Provided Menu Object, and any Key Objects.

Parameters

ScaleFactor	

Reimplemented in UAccessibilityAddNodeContextMenu, and UAccessibilityGraphEditorContext.

Definition at line 87 of file ContextMenuObject.h.

```
00087 {};
```

4.58.3.11 SetMenu()

```
virtual void UPhraseTreeContextMenuObject::SetMenu (
             TSharedRef< IMenu > InMenu )  [inline], [virtual]
```

Sets the Menu Object for this Context Menu.

Parameters

InMenu	

Definition at line 78 of file ContextMenuObject.h.

```
00079     {
00080         Menu = InMenu;
00081     }
```

4.58.3.12 Tick()

```
virtual bool UPhraseTreeContextMenuObject::Tick (
             float DeltaTime )  [inline], [virtual]
```

Definition at line 38 of file ContextMenuObject.h.

```
00038 { return true; };
```

4.58.4 Member Data Documentation

4.58.4.1 Menu

```
TWeakPtr<IMenu> UPhraseTreeContextMenuObject::Menu
```

The Menu Object.

Definition at line 111 of file ContextMenuObject.h.

4.58.4.2 Window

```
TWeakPtr<SWindow> UPhraseTreeContextMenuObject::Window
```

The Menu's Window.

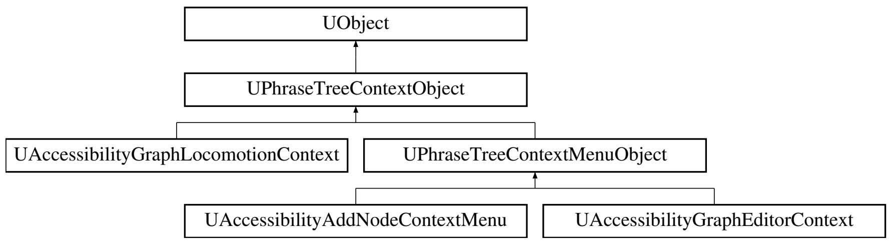
Definition at line 116 of file ContextMenuObject.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/ContextMenuObject.h
- Source/OpenAccessibilityCommunication/Private/PhraseTree/Containers/ContextMenuObject.cpp

4.59 UPhraseTreeContextObject Class Reference

Inheritance diagram for UPhraseTreeContextObject:


UObject
UPhraseTreeContextObject
UAccessibilityGraphLocomotionContext
UPhraseTreeContextMenuObject
UAccessibilityAddNodeContextMenu
UAccessibilityGraphEditorContext

Public Member Functions

- virtual bool Close ()
- void SetContextRootNode (TSharedRef< FPhraseNode > InRootNode)

  Sets the Root Node In The Phrase Tree For This Context Objects.
- TSharedPtr< FPhraseNode > GetContextRoot ()

  Gets the Root Node For This Context.
- const bool GetIsActive ()

  Is the Context Still Active.

Protected Attributes

- bool bIsActive = true

  Is the Context Object Still Active.
- TWeakPtr< FPhraseNode > ContextRoot

  The Root Node In The Phrase Tree (The Origin of the Context). Allowing for Propagation based on Context Root.

4.59.1 Detailed Description

Definition at line 12 of file ContextObject.h.

4.59.2 Constructor & Destructor Documentation

4.59.2.1 UPhraseTreeContextObject()

```
UPhraseTreeContextObject::UPhraseTreeContextObject ( )  [inline]
```

Definition at line 18 of file ContextObject.h.

```
        : UObject()
    {

    }
```

4.59.2.2 ∼UPhraseTreeContextObject()

```
UPhraseTreeContextObject::∼UPhraseTreeContextObject ( )  [inline]
```

Definition at line 24 of file ContextObject.h.

```
    {

    }
```

4.59.3 Member Function Documentation

4.59.3.1 Close()

```
virtual bool UPhraseTreeContextObject::Close ( )  [inline], [virtual]
```

Reimplemented in UAccessibilityAddNodeContextMenu, UAccessibilityGraphEditorContext, and UPhraseTreeContextMenuObject.

Definition at line 29 of file ContextObject.h.

```
{ return true; }
```

4.59.3.2 GetContextRoot()

```
TSharedPtr< FPhraseNode > UPhraseTreeContextObject::GetContextRoot ( )  [inline]
```

Gets the Root Node For This Context.

Returns

Definition at line 44 of file ContextObject.h.

```
    {
        return ContextRoot.Pin();
    }
```

4.59.3.3 GetIsActive()

```
const bool UPhraseTreeContextObject::GetIsActive ( )  [inline]
```

Is the Context Still Active.

Returns

Definition at line 53 of file ContextObject.h.

```
    {
        return bIsActive;
    }
```

4.59.3.4 SetContextRootNode()

```
void UPhraseTreeContextObject::SetContextRootNode (
             TSharedRef< FPhraseNode > InRootNode )  [inline]
```

Sets the Root Node In The Phrase Tree For This Context Objects.

Parameters

InRootNode	

Definition at line 35 of file ContextObject.h.

```
    {
        ContextRoot = InRootNode;
    }
```

4.59.4 Member Data Documentation

4.59.4.1 bIsActive

```
bool UPhraseTreeContextObject::bIsActive = true  [protected]
```

Is the Context Object Still Active.

Definition at line 63 of file ContextObject.h.

4.59.4.2 ContextRoot

```
TWeakPtr<FPhraseNode> UPhraseTreeContextObject::ContextRoot  [protected]
```

The Root Node In The Phrase Tree (The Origin of the Context). Allowing for Propagation based on Context Root.

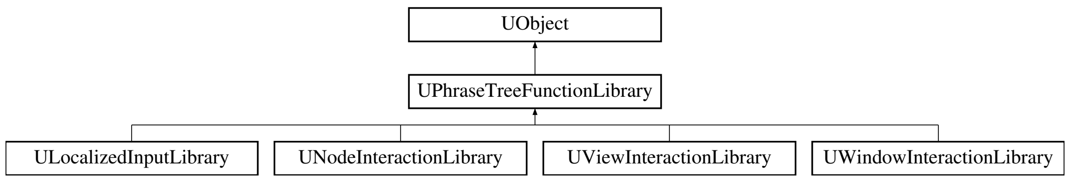
Definition at line 69 of file ContextObject.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/ContextObject.h

4.60 UPhraseTreeFunctionLibrary Class Reference

Inheritance diagram for UPhraseTreeFunctionLibrary:



Public Member Functions

- virtual void BindBranches (TSharedRef< FPhraseTree > PhraseTree)

4.60.1 Detailed Description

Definition at line 27 of file PhraseTreeFunctionLibrary.h.

4.60.2 Member Function Documentation

4.60.2.1 BindBranches()

```
virtual void UPhraseTreeFunctionLibrary::BindBranches (
            TSharedRef< FPhraseTree > PhraseTree )  [inline], [virtual]
```

Reimplemented in ULocalizedInputLibrary, UNodeInteractionLibrary, UViewInteractionLibrary, and UWindowInteractionLibrary.

Definition at line 33 of file PhraseTreeFunctionLibrary.h.

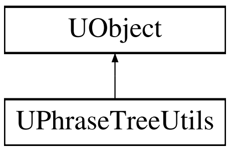
```
00033 {};
```

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

- Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseTreeFunctionLibrary.h

4.61 UPhraseTreeUtils Class Reference

Inheritance diagram for UPhraseTreeUtils:



Public Member Functions

- void RegisterFunctionLibrary (UPhraseTreeFunctionLibrary ∗LibraryToRegister)
- void SetPhraseTree (TSharedRef< FPhraseTree > NewPhraseTree)

Protected Attributes

- TArray< UPhraseTreeFunctionLibrary ∗ > RegisteredLibraries
- TWeakPtr< FPhraseTree > PhraseTree

4.61.1 Detailed Description

Definition at line 12 of file PhraseTreeUtils.h.

4.61.2 Constructor & Destructor Documentation

4.61.2.1 UPhraseTreeUtils()

```
UPhraseTreeUtils::UPhraseTreeUtils ( )
```

Definition at line 5 of file PhraseTreeUtils.cpp.

```
00006 {
00007
00008 }
```

4.61.2.2 ∼UPhraseTreeUtils()

```
UPhraseTreeUtils::∼UPhraseTreeUtils ( ) [virtual]
```

Definition at line 10 of file PhraseTreeUtils.cpp.

```
00011 {
00012
00013 }
```

4.61.3 Member Function Documentation

4.61.3.1 RegisterFunctionLibrary()

```
void UPhraseTreeUtils::RegisterFunctionLibrary (
            UPhraseTreeFunctionLibrary * LibraryToRegister )
```

Registers the provided Phrase Tree Function Library.

Parameters

LibraryToRegister	The Phrase Tree Function Library to Register.

Definition at line 15 of file PhraseTreeUtils.cpp.

```
{
    TSharedPtr<FPhraseTree> PhraseTreeSP = PhraseTree.Pin();
    if (!PhraseTreeSP.IsValid())
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("Cannot Register Phrase Tree Function Library
       Due To InValid Phrase Tree Reference."));
        return;
    }

    // For some reason this needs to be told directly to be kept alive,
    // even though it is a UPROPERTY TArray and should be kept alive by the UObject system.
    LibraryToRegister->AddToRoot();
    LibraryToRegister->BindBranches(PhraseTreeSP.ToSharedRef());
}
```

4.61.3.2 SetPhraseTree()

```
void UPhraseTreeUtils::SetPhraseTree (
              TSharedRef< FPhraseTree > NewPhraseTree )  [inline]
```

Sets the Phrase Tree Reference used for Registering Phrase Tree Function Libraries.

Parameters

NewPhraseTree	Reference to the Phrase Tree to use.

Definition at line 34 of file PhraseTreeUtils.h.

```
    {
        this->PhraseTree = NewPhraseTree;
    }
```

4.61.4 Member Data Documentation

4.61.4.1 PhraseTree

```
TWeakPtr<FPhraseTree> UPhraseTreeUtils::PhraseTree  [protected]
```

Weak Pointer to the Current Phrase Tree Instance used in Registering the Phrase Tree Function Libraries.

Definition at line 51 of file PhraseTreeUtils.h.

4.61.4.2 RegisteredLibraries

```
TArray<UPhraseTreeFunctionLibrary*> UPhraseTreeUtils::RegisteredLibraries  [protected]
```

An Array of all the Registered Phrase Tree Function Libraries.

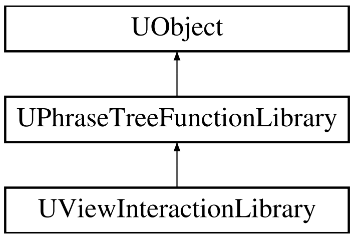
Definition at line 45 of file PhraseTreeUtils.h.

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

- Source/OpenAccessibilityCommunication/Public/PhraseTreeUtils.h
- Source/OpenAccessibilityCommunication/Private/PhraseTreeUtils.cpp

4.62 UViewInteractionLibrary Class Reference

Inheritance diagram for UViewInteractionLibrary:



Public Member Functions

- UViewInteractionLibrary (const FObjectInitializer &ObjectInitializer)
- void BindBranches (TSharedRef< FPhraseTree > PhraseTree) override
- void MoveViewport (FParseRecord &Record)
- void ZoomViewport (FParseRecord &Record)
- void IndexFocus (FParseRecord &Record)

4.62.1 Detailed Description

Definition at line 12 of file ViewInteractionLibrary.h.

4.62.2 Constructor & Destructor Documentation

4.62.2.1 UViewInteractionLibrary()

```
UViewInteractionLibrary::UViewInteractionLibrary (
            const FObjectInitializer & ObjectInitializer )
```

Definition at line 12 of file ViewInteractionLibrary.cpp.

```
00013     : Super(ObjectInitializer)
00014 {
00015
00016 }
```

4.62.2.2 ∼UViewInteractionLibrary()

```
UViewInteractionLibrary::∼UViewInteractionLibrary ( )  [virtual]
```

Definition at line 18 of file ViewInteractionLibrary.cpp.

```
{

}
```

4.62.3 Member Function Documentation

4.62.3.1 BindBranches()

```
void UViewInteractionLibrary::BindBranches (
            TSharedRef< FPhraseTree > PhraseTree )  [override], [virtual]
```

Binds Branches originating from this Library onto the provided Phrase Tree.

Parameters

PhraseTree	Reference to the PhraseTree to Bind this Library to.

Reimplemented from UPhraseTreeFunctionLibrary.

Definition at line 23 of file ViewInteractionLibrary.cpp.

```
{
    PhraseTree->BindBranch(
        MakeShared<FPhraseNode>(TEXT("VIEW"),
        TPhraseNodeArray {

            MakeShared<FPhraseNode>(TEXT("MOVE"),
            TPhraseNodeArray {

                MakeShared<FPhrase2DDirectionalInputNode>(TEXT("DIRECTION"),
                TPhraseNodeArray {

                    MakeShared<FPhraseInputNode<int32»(TEXT("AMOUNT"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
       &UViewInteractionLibrary::MoveViewport))

                    })

                })

            }),

            MakeShared<FPhraseNode>(TEXT("ZOOM"),
            TPhraseNodeArray {

                MakeShared<FPhrase2DDirectionalInputNode>(TEXT("DIRECTION"),
                TPhraseNodeArray {

                    MakeShared<FPhraseInputNode<int32»(TEXT("AMOUNT"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
       &UViewInteractionLibrary::ZoomViewport))

                    })

                })

            }),

            MakeShared<FPhraseNode>(TEXT("FOCUS"),
            TPhraseNodeArray {

                MakeShared<FPhraseInputNode<int32>>(TEXT("INDEX"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UViewInteractionLibrary::IndexFocus))

                })

            })

        })
    );
}
```

4.62.3.2 IndexFocus()

```
void UViewInteractionLibrary::IndexFocus (
            FParseRecord & Record )
```

Phrase Event for Focusing on the Active Viewports Indexed Item, if one is apparent.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 165 of file ViewInteractionLibrary.cpp.

```
{
    GET_ACTIVE_TAB(ActiveTab)

    FString TabType = ActiveTab->GetTypeAsString();

    UParseIntInput* IndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("INDEX"));
    if (IndexInput == nullptr)
        return;

    if (TabType == "SGraphEditor")
    {
        TSharedPtr<SGraphEditor> GraphEditor = StaticCastSharedPtr<SGraphEditor>(ActiveTab);
        if (!GraphEditor.IsValid())
            return;

        TSharedRef<FAssetAccessibilityRegistry> AssetRegistry = GetAssetRegistry();

        TSharedRef<FGraphIndexer> GraphIndexer = AssetRegistry->GetGraphIndexer(GraphEditor->GetCurrentGraph());

        UEdGraphNode* Node = GraphIndexer->GetNode(IndexInput->GetValue());
        if (Node == nullptr)
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("IndexFocus: INVALID INDEX INPUT"))
            return;
        }

        GraphEditor->JumpToNode(Node);
    }

    // Further ViewportS Specific Implementation Here
}
```

4.62.3.3 MoveViewport()

```
void UViewInteractionLibrary::MoveViewport (
            FParseRecord & Record )
```

Phrase Event for Moving the Active Viewport.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 79 of file ViewInteractionLibrary.cpp.

```
                                                             {
    GET_ACTIVE_TAB(ActiveTab)

    FString TabType = ActiveTab->GetTypeAsString();

    UParseEnumInput* DirectionInput = Record.GetPhraseInput<UParseEnumInput>(TEXT("DIRECTION"));
    UParseIntInput* AmountInput = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (DirectionInput == nullptr || AmountInput == nullptr)
        return;

    if (TabType == "SGraphEditor")
    {
        TSharedPtr<SGraphEditor> GraphEditor = StaticCastSharedPtr<SGraphEditor>(ActiveTab);

        FVector2D ViewLocation;
        float ZoomAmount;
        GraphEditor->GetViewLocation(ViewLocation, ZoomAmount);

        switch (EPhrase2DDirectionalInput(DirectionInput->GetValue()))
        {
            case EPhrase2DDirectionalInput::UP:
                ViewLocation.Y -= AmountInput->GetValue();
                break;

            case EPhrase2DDirectionalInput::DOWN:
                ViewLocation.Y += AmountInput->GetValue();
                break;

            case EPhrase2DDirectionalInput::LEFT:
                ViewLocation.X -= AmountInput->GetValue();
                break;

            case EPhrase2DDirectionalInput::RIGHT:
                ViewLocation.X += AmountInput->GetValue();
                break;

            default:
                UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("MoveViewport: INVALID DIRECTION
       INPUT"));
                return;
        }

        GraphEditor->SetViewLocation(ViewLocation, ZoomAmount);
    }

    // Further Viewport Implementation Here
}
```

4.62.3.4 ZoomViewport()

```
void UViewInteractionLibrary::ZoomViewport (
            FParseRecord & Record )
```

Phrase Event for Zooming the Active Viewport.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 126 of file ViewInteractionLibrary.cpp.

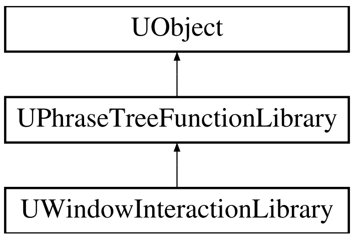
```
{
    GET_ACTIVE_TAB(ActiveTab)

    FString TabType = ActiveTab->GetTypeAsString();

    UParseEnumInput* DirectionInput = Record.GetPhraseInput<UParseEnumInput>(TEXT("DIRECTION"));
    UParseIntInput* AmountInput = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (DirectionInput == nullptr || AmountInput == nullptr)
        return;

    if (TabType == "SGraphEditor")
    {
        TSharedPtr<SGraphEditor> GraphEditor = StaticCastSharedPtr<SGraphEditor>(ActiveTab);

        FVector2D ViewLocation;
        float ZoomAmount;
        GraphEditor->GetViewLocation(ViewLocation, ZoomAmount);

        switch (EPhrase2DDirectionalInput(DirectionInput->GetValue())) {
            case EPhrase2DDirectionalInput::UP:
                ZoomAmount += AmountInput->GetValue();
                break;

            case EPhrase2DDirectionalInput::DOWN:
                ZoomAmount -= AmountInput->GetValue();
                break;

            default:
                UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("ZoomViewport: INVALID DIRECTION
INPUT"));
                return;
        }

        GraphEditor->SetViewLocation(ViewLocation, ZoomAmount);
    }

    // Further Viewport Specific Implementation Here
}
```

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

- Source/OpenAccessibility/Public/PhraseEvents/ViewInteractionLibrary.h
- Source/OpenAccessibility/Private/PhraseEvents/ViewInteractionLibrary.cpp

4.63 UWindowInteractionLibrary Class Reference

Inheritance diagram for UWindowInteractionLibrary:



Public Member Functions

- UWindowInteractionLibrary (const FObjectInitializer &ObjectInitializer)
- void BindBranches (TSharedRef< FPhraseTree > PhraseTree) override
- void CloseActiveWindow (FParseRecord &Record)
- void SelectToolBarItem (FParseRecord &Record)

Protected Attributes

- class UAccessibilityWindowToolbar ∗ WindowToolBar

4.63.1 Detailed Description

Definition at line 12 of file WindowInteractionLibrary.h.

4.63.2 Constructor & Destructor Documentation

4.63.2.1 UWindowInteractionLibrary()

```
UWindowInteractionLibrary::UWindowInteractionLibrary (
            const FObjectInitializer & ObjectInitializer )
```

Definition at line 10 of file WindowInteractionLibrary.cpp.

```
    : Super(ObjectInitializer)
{
    WindowToolBar = NewObject<UAccessibilityWindowToolbar>();
}
```

4.63.2.2 ∼UWindowInteractionLibrary()

```
UWindowInteractionLibrary::∼UWindowInteractionLibrary ( )  [virtual]
```

Definition at line 16 of file WindowInteractionLibrary.cpp.

```
{

}
```

4.63.3 Member Function Documentation

4.63.3.1 BindBranches()

```
void UWindowInteractionLibrary::BindBranches (
            TSharedRef< FPhraseTree > PhraseTree )  [override], [virtual]
```

Binds Branches originating from this Library onto the provided Phrase Tree.

Parameters

PhraseTree	Reference to the PhraseTree to Bind this Library to.

Reimplemented from UPhraseTreeFunctionLibrary.

Definition at line 21 of file WindowInteractionLibrary.cpp.

```
{
    PhraseTree->BindBranches(
        TPhraseNodeArray{

            MakeShared<FPhraseNode>(TEXT("WINDOW"),
            TPhraseNodeArray{

                MakeShared<FPhraseNode>(TEXT("CLOSE"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
       &UWindowInteractionLibrary::CloseActiveWindow))

                }),

            }),

            MakeShared<FPhraseNode>(TEXT("TOOLBAR"),
            TPhraseNodeArray {

                MakeShared<FPhraseInputNode<int32»(TEXT("ITEM_INDEX"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
       &UWindowInteractionLibrary::SelectToolBarItem))

                })

            })

        }
    );
}
```

4.63.3.2 CloseActiveWindow()

```
void UWindowInteractionLibrary::CloseActiveWindow (
              FParseRecord & Record )
```

Closes the Top Most Active Window, if it is not the Root Application Window.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 54 of file WindowInteractionLibrary.cpp.

```
                                                                      {
    FSlateApplication& SlateApp = FSlateApplication::Get();
    if (!SlateApp.CanDisplayWindows())
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("CloseActiveWindow: Slate Application
       cannot display windows."));
        return;
    }

    TSharedPtr<SWindow> ActiveWindow = SlateApp.GetActiveTopLevelWindow();
    if (!ActiveWindow.IsValid())
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("CloseActiveWindow: No Active Window
       Found."));
        return;
    }

    TSharedPtr<SWindow> RootWindow = FGlobalTabmanager::Get()->GetRootWindow();
    if (ActiveWindow->IsVisible() && ActiveWindow != RootWindow)
    {
        ActiveWindow->RequestDestroyWindow();
    }
}
```

4.63.3.3 SelectToolBarItem()

```
void UWindowInteractionLibrary::SelectToolBarItem (
            FParseRecord & Record )
```

Selects the Item from the Active Windows ToolBar.

Parameters

Record	The Parse Record accumulated until this Event.

Definition at line 76 of file WindowInteractionLibrary.cpp.

```
{
    UParseIntInput* ItemIndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("ITEM_INDEX"));
    if (ItemIndexInput == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("SelectToolBarItem: No Item Index
    Found."));
        return;
    }

    WindowToolBar->SelectToolbarItem(ItemIndexInput->GetValue());
}
```

4.63.4 Member Data Documentation

4.63.4.1 WindowToolBar

```
class UAccessibilityWindowToolbar* UWindowInteractionLibrary::WindowToolBar  [protected]
```

Definition at line 58 of file WindowInteractionLibrary.h.

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

- Source/OpenAccessibility/Public/PhraseEvents/WindowInteractionLibrary.h
- Source/OpenAccessibility/Private/PhraseEvents/WindowInteractionLibrary.cpp

4.64 OpenAccessibilityPy.WhisperInterface.WhisperInterface Class Reference

Public Member Functions

- def __init__ (self, str model_name="distil-small.en", str device="auto", int cpu_threads=4, int transcribe_↩ workers=2, str compute_type="default")
- def __del__ (self)
- def process_file_from_dir (self, str filepath)
- tuple[list[Segment], dict] process_audio_buffer (self, np.ndarray audio_buffer)

Public Attributes

- whisper_model
- beam_size

4.64.1 Detailed Description

```
Interface Class for Interacting with the CTranslate2 Faster Whisper Model.
```

Definition at line 13 of file WhisperInterface.py.

4.64.2 Constructor & Destructor Documentation

4.64.2.1 __init__()

```
def OpenAccessibilityPy.WhisperInterface.WhisperInterface.__init__ (
                 self,
                str  model_name = "distil-small.en",
                str  device = "auto",
                int  cpu_threads = 4,
                int  transcribe_workers = 2,
                str  compute_type = "default" )
```

```
Constructor of Whisper Interface Class

Args:
    model_name (str, optional): Hugging-Face Model Specifier for CTranslate Compatible Models. Defaults to "di
    device (str, optional): Device host for the Whisper Model (Can be "auto", "cpu", "cuda"). Defaults to "aut
    cpu_threads (int, optional): Amount of CPU Threads to use, if Hosting the Model on a CPU. Defaults to 4.
    transcribe_workers (int, optional): Amount of Thread Workers for Audio Transcription. Defaults to 2.
    compute_type (str, optional): Data Structure for Whisper Compute. Defaults to "default".
```

Definition at line 16 of file WhisperInterface.py.

```
    ):
        """Constructor of Whisper Interface Class

        Args:
            model_name (str, optional): Hugging-Face Model Specifier for CTranslate Compatible Models. Defaults to "distil-small.en".
            device (str, optional): Device host for the Whisper Model (Can be "auto", "cpu", "cuda"). Defaults to "auto".
            cpu_threads (int, optional): Amount of CPU Threads to use, if Hosting the Model on a CPU. Defaults to 4.
            transcribe_workers (int, optional): Amount of Thread Workers for Audio Transcription. Defaults to 2.
            compute_type (str, optional): Data Structure for Whisper Compute. Defaults to "default".
        """

        # Whisper Focused Variables
        self.whisper_model = WhisperModel(
            model_name,
            device=device,
            compute_type=compute_type,
            num_workers=transcribe_workers,
            cpu_threads=cpu_threads,
            local_files_only=True,
        )
        self.beam_size = 5

```

4.64.2.2 __del__()

```
def OpenAccessibilityPy.WhisperInterface.WhisperInterface.__del__ (
              self )
```

```
Destructor of Whisper Interface Class.
```

Definition at line 45 of file WhisperInterface.py.

```
00045     def __del__(self):
00046         """Destructor of Whisper Interface Class."""
00047
00048         del self.whisper_model
00049
```

4.64.3 Member Function Documentation

4.64.3.1 process_audio_buffer()

```
tuple[list[Segment], dict] OpenAccessibilityPy.WhisperInterface.WhisperInterface.process_↩
audio_buffer (
              self,
             np.ndarray  audio_buffer )
```

```
Transcribes an NDArray AudioBuffer.

Args:
    audio_buffer (np.ndarray): AudioBuffer to Transcribe.

Returns:
    tuple[list[Segment], dict]: Tuple Containing a List of Transcription Segments and a Dictionary of Collecte
```

Definition at line 80 of file WhisperInterface.py.

```
00082     ) -> tuple[list[Segment], dict]:
00083         """Transcribes an NDArray AudioBuffer.
00084
00085         Args:
00086             audio_buffer (np.ndarray): AudioBuffer to Transcribe.
00087
00088         Returns:
00089             tuple[list[Segment], dict]: Tuple Containing a List of Transcription Segments and a
       Dictionary of Collected Metadata.
00090         """
00091
00092         segments, info = self.whisper_model.transcribe(
00093             audio_buffer,
00094             beam_size=self.beam_size,
00095             language="en",
00096         )
00097
00098         Log(
00099             f"WhisperInterface || Detected Language: {info.language} | Probability:
       {info.language_probability} | Duration: {info.duration}"
00100         )
00101
00102         collected_metadata = {
00103             "duration": info.duration,
00104             "language": info.language,
00105             "language_probability": info.language_probability,
00106         }
00107
00108         return list(segments), collected_metadata
```

4.64.3.2 process_file_from_dir()

```
def OpenAccessibilityPy.WhisperInterface.WhisperInterface.process_file_from_dir (
              self,
             str filepath )
```

```
Transcribes an Audio Files from a Given WAV File Path.

Args:
    filepath (str): Path to the Audio Files to Transcribe.

Returns:
    A List of Segments containing the Transcribed Text and their Time Stamps.
```

Definition at line 50 of file WhisperInterface.py.

```
    def process_file_from_dir(self, filepath: str):
        """Transcribes an Audio Files from a Given WAV File Path.

        Args:
            filepath (str): Path to the Audio Files to Transcribe.

        Returns:
            A List of Segments containing the Transcribed Text and their Time Stamps.
        """

        segments, info = self.whisper_model.transcribe(
            filepath,
            beam_size=self.beam_size,
            language="en",
            prepend_punctuations="",
            append_punctuations="",
            vad_filter=True,
        )

        Log(
            f"WhisperInterface | Detected Language: {info.language} | Probability:
       {info.language_probability} | Duration: {info.duration}"
        )

        for segment in segments:
            Log(
                f"WhisperInterface | Segment : {segment.text} | Start: {segment.start} | End:
       {segment.end}"
            )

        return list(segments)

```

4.64.4 Member Data Documentation

4.64.4.1 beam_size

`OpenAccessibilityPy.WhisperInterface.WhisperInterface.beam_size`

Definition at line 43 of file WhisperInterface.py.

4.64.4.2 whisper_model

`OpenAccessibilityPy.WhisperInterface.WhisperInterface.whisper_model`

Definition at line 35 of file WhisperInterface.py.

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

- Content/Python/OpenAccessibilityPy/WhisperInterface.py

Chapter 5

File Documentation

5.1 init_unreal.py

```python
import unreal

import subprocess
import pkg_resources
from os import path


def get_requirements(requirements_dir: str) -> list[str]:
    with open(
        path.join(requirements_dir, "requirements.txt"), "r"
    ) as requirements_file:
        return [line.strip() for line in requirements_file.readlines()]


def is_dependency_satisfied(dependency: str) -> bool:
    try:
        pkg_resources.require(dependency)
        return True
    except:
        return False


def install_dependencies(deps_to_install: list):

    unreal.log_warning(
        f"|| OpenAccessibility Python || Installing Dependencies: {deps_to_install} ||"
    )

    with unreal.ScopedSlowTask(
        len(deps_to_install),
        "OpenAccessibility Installing Python Dependencies",
        enabled=True,
    ) as install_ui:
        process = subprocess.Popen(
            [
                unreal.get_interpreter_executable_path(),
                "-m",
                "pip",
                "install",
            ]
            + deps_to_install,
            shell=True,
            stdin=subprocess.PIPE,
            stdout=subprocess.PIPE,
            stderr=None,
        )

        while process.poll() is None:
            install_ui.enter_progress_frame(
                0, process.stdout.readline().decode("utf-8")
            )

        process.wait()




unreal.log("|| OpenAccessibility Python || Initializing ||")
```

```



# Verify Required Dependencies

missing_deps = [
    dep
    for dep in get_requirements(path.dirname(path.realpath(__file__)))
    if not is_dependency_satisfied(dep)
]

if missing_deps:
    install_dependencies(missing_deps)



# Initialize the Python Runtime

unreal.log("|| OpenAccessibility Python || Starting Python Runtime ||")

import OpenAccessibilityPy as OAPy

# Run Utilities for Better Project Runtime.

# Helps Circumvent CUDA and CUDNN Issues
# during the inference process with the Whisper Model.
# OAPy.forward_CUDA_CUDNN_to_path()

# Initialize the Runtime
OpenAccessibilityPy = OAPy.OpenAccessibilityPy()


```

5.2 old_init_unreal.py

```
import unreal

import subprocess
import pkg_resources

unreal.log("|| OpenAccessibility Python || Initializing")

# Dependencies of the Project
DEPS = ["faster-whisper", "pyzmq", "av"]
installed = {pkg for pkg in pkg_resources.working_set}

missing_dependencies = DEPS - installed

if missing_dependencies:
    unreal.log_warning(
        "|| OpenAccessibility Python || Missing Dependencies Detected ||"
    )

    with unreal.ScopedSlowTask(
        len(missing_dependencies), "OpenAccessibilty Installing Python Dependencies"
    ) as slow_task:

        # Create a Dialog for UI Feedback
        slow_task.make_dialog(can_cancel=False)

        for depNum, depName in enumerate(missing_dependencies):
            unreal.log_warning(
                f"|| OpenAccessibility Python || Installing {depName} ||"
            )

            slow_task.enter_progress_frame(
                0.5,
                f"Installing Dependency {depNum} / {len(missing_dependencies)}: {depName}",
            )

            process = subprocess.Popen(
                [
                    unreal.get_interpreter_executable_path(),
                    "-m",
                    "pip",
                    "install",
                    depName,
                ],
                shell=True,
            )

            while process.poll() is None:

```

```python
                slow_task.enter_progress_frame(
                    0.5,
                    f"Installed Dependency {depNum} / {len(missing_dependencies)}: {depName}",
                )
else:
    unreal.log(
        "|| OpenAccessibility Python || All Dependencies are already installed ||"
    )

import OpenAccessibilityPy as OAPy

unreal.log("|| OpenAccessibility Python || Initializing Python Runtime ||")

# Run Utilities for Better Project Library Initialization

# Helps Circumvent CUDA and CUDNN Issues
# when using the Whisper Model
OAPy.forward_CUDA_CUDNN_to_path()

# Initialize the Python Runtime
OpenAccessibilityPy = OAPy.OpenAccessibilityPy()
```

5.3 __init__.py

```python
import unreal as ue
import zmq
import numpy as np
from gc import collect as gc_collect

from concurrent.futures import ThreadPoolExecutor as ThreadPool

from .CommunicationServer import CommunicationServer
from .WhisperInterface import WhisperInterface
from .Audio import AudioResampler
from .Logging import Log, LogLevel

from .LibUtils import (
    get_filtered_path_list,
    get_child_directories,
    append_paths_to_library_path,
)


def forward_CUDA_CUDNN_to_path():
    """
    Forces any found CUDA and CUDNN Paths to the System Path.

    This is useful for circumventing issues with CUDA and CUDNN not being found on the embedded interpreter.
    Not always needed, but useful for some systems.
    """

    filtered_path_list = get_filtered_path_list(["CUDA", "CUDNN"])

    for path in filtered_path_list:
        append_paths_to_library_path(get_child_directories(path, depth=1))


class OpenAccessibilityPy:
    """Python Runtime Class for Open Accessbility Plugin"""

    def __init__(
        self,
        # General Runtime Specifics
        worker_count: int = 2,
        # Whisper Specifics
        whisper_model: str = "distil-small.en",
        device: str = "auto",
        compute_type: str = "default",
        # Communication Specifics
        poll_timeout: int = 10,
    ):
        """Constructor of Python Runtime Class for Open Accessibility Plugin

        Args:
            worker_count (int, optional): Amount of Thread Workers for Audio Transcription. Defaults to 2.
            whisper_model (str, optional): Hugging-Face Model Specifier for CTranslate Compatible Models. Defaults to "distil-small.en".
            device (str, optional): Device host for the Whisper Model (Can be "auto", "cpu", "cuda"). Defaults to "auto".
            compute_type (str, optional): Data Structure for Whisper Compute. Defaults to "default".
            poll_timeout (int, optional): Amount of time (ms) for event polling on the Transcription Socket. Defaults to 10.
        """

        self.worker_pool = ThreadPool(
            max_workers=worker_count, thread_name_prefix="TranscriptionWorker"
        )

        self.whisper_interface = WhisperInterface(
            model_name=whisper_model,
            device=device,
            compute_type=compute_type,
            transcribe_workers=worker_count,
        )
        self.com_server = CommunicationServer(
            send_socket_type=zmq.PUSH,
            recv_socket_type=zmq.PULL,
            poll_timeout=poll_timeout,
        )
        self.audio_resampler = AudioResampler(target_sample_rate=16000)

        self.tick_handle = ue.register_slate_post_tick_callback(self.Tick)

        self.pyshutdown_handle = ue.register_python_shutdown_callback(self.Shutdown)

    def __del__(self):
        """Destructor of Python Runtime Class for Open Accessibility Plugin"""

        self.Shutdown()

    def Tick(self, delta_time: float):
        """Tick Callback for Unreal Engine Slate Post Tick.

        Detecting Incoming Transcription Requests and Handling them, through the Worker Pool.

        Args:
            delta_time (float): Time since last tick
        """

        if self.com_server.EventOccured():
            Log("Event Occured")

            message, metadata = self.com_server.ReceiveNDArrayWithMeta(dtype=np.float32)

            self.worker_pool.submit(self.HandleTranscriptionRequest, message, metadata)

    def HandleTranscriptionRequest(
        self, recv_message: np.ndarray, metadata: dict = None
    ):
        """Handles Incoming Transcription Requests

        Takes the Incoming AudioBuffer, Resamples it to 16kHz and Transcribes it using Whisper.

        Args:
            recv_message (np.ndarray): ndarray of the incoming audio buffer.
            metadata (dict, optional): metadata of the incoming audio buffer, if any is recieved.
Defaults to None.
        """

        Log(
            f"Handling Transcription Request | Message: {recv_message} | Size: {recv_message.size} |
Shape: {recv_message.shape}"
        )

        sample_rate = metadata.get("sample_rate", 48000)
        num_channels = metadata.get("num_channels", 2)

        message_ndarray = self.audio_resampler.resample(
            recv_message, sample_rate, num_channels
        )

        trans_segments, trans_metadata = self.whisper_interface.process_audio_buffer(
            message_ndarray
        )

        encoded_segments = [
            transcription.text.encode() for transcription in trans_segments
        ]

        Log(f"Encoded Segments: {encoded_segments}")

        if len(encoded_segments) > 0:
            try:
                self.com_server.SendMultipartWithMeta(
                    message=encoded_segments, meta=trans_metadata
                )

            except:
                Log("Error Sending Encoded Transcription Segments", LogLevel.ERROR)
```

```python

        else:
            Log("No Transcription Segments Returned", LogLevel.WARNING)

    def Shutdown(self):
        """Shutsdown the Python Runtime Components, and Forces a Garbage Collection."""

        if self.tick_handle:
            ue.unregister_slate_post_tick_callback(self.tick_handle)
            del self.tick_handle

        if self.worker_pool:
            self.worker_pool.shutdown(wait=False, cancel_futures=True)
            del self.worker_pool

        if self.audio_resampler:
            del self.audio_resampler

        if self.com_server:
            del self.com_server

        if self.whisper_interface:
            del self.whisper_interface

        # Force a Garbage Collection
        gc_collect()
```

5.4 __main__.py

```python
import numpy as np
from zmq import PUSH as zmq_PUSH, PULL as zmq_PULL

from faster_whisper.transcribe import decode_audio

from CommunicationServer import CommunicationServer
from WhisperInterface import WhisperInterface
from Audio import AudioResampler
import LibUtils

from Logging import Log, LogLevel


PERFORM_COMPARE = False


def PlotAudioBuffers(
    recv_audio_buffer: np.ndarray,
    decoded_audio_buffer: np.ndarray,
    name: str = "BufferComparison",
):
    """
    Plots the received audio buffer and the decoded audio buffer to compare the two.
    """

    try:
        import matplotlib as mpl
        from matplotlib import pyplot as plt

        mpl.interactive(False)

        fig, axs = plt.subplots(3)

        axs[0].plot(recv_audio_buffer)

        axs[1].plot(decoded_audio_buffer)

        axs[2].plot(recv_audio_buffer)
        axs[2].plot(decoded_audio_buffer)
        axs[2].set_title("Buffer Comparison")

        fig.savefig(
            f"D:/dev/Unreal Engine/AccessibilityProject/Saved/Debug/OpenAccessibility/{name}.png",
            dpi=300,
        )

        fig.clear()

    except Exception as e:
        Log(f"Error Plotting Audio Buffers: {e}", LogLevel.ERROR)


def main():

    whisper_interface = WhisperInterface("distil-small.en")
    com_server = CommunicationServer(
        send_socket_type=zmq_PUSH, recv_socket_type=zmq_PULL, poll_timeout=10
    )
    audio_resampler = AudioResampler(target_sample_rate=16000)

    should_run = True

    print("Starting Run Loop")

    while should_run:

        if com_server.EventOccured():
            Log("Event Occured")

            recv_message, metadata = com_server.ReceiveNDArrayWithMeta()

            message_ndarray: np.ndarray = np.frombuffer(recv_message, dtype=np.float32)

            sample_rate = metadata.get("sample_rate", 48000)
            num_channels = metadata.get("num_channels", 1)

            if PERFORM_COMPARE:
                decoded_ndarray = decode_audio(
                    "D:/dev/Unreal
Engine/AccessibilityProject/Saved/BouncedWavFiles/OpenAccessibility/Audioclips/Captured_User_Audio.wav",
                    sampling_rate=16000,
                )

                PlotAudioBuffers(message_ndarray, decoded_ndarray)

                isSame = np.array_equal(message_ndarray, decoded_ndarray)
                # isClose = np.allclose(message_ndarray, decoded_ndarray)

                # difference = np.subtract(message_ndarray, decoded_ndarray)

                Log(
                    f"Recieved Buffer | {message_ndarray} | Shape: {message_ndarray.shape}"
                )

                Log(
                    f"Decoded Buffer | {decoded_ndarray} | Shape: {decoded_ndarray.shape}"
                )
                Log(f"Comparisons | Is Same: {isSame}")

            # Apply Resampling to the Audio Buffer, to match samplerate of 16000Hz
            message_ndarray = audio_resampler.resample(message_ndarray, sample_rate)

            if PERFORM_COMPARE:
                PlotAudioBuffers(
                    message_ndarray, decoded_ndarray, name="ResampledBufferComparison"
                )

            transcription_segments, metadata = whisper_interface.process_audio_buffer(
                message_ndarray
            )

            encoded_segments = [
                transcription.text.encode() for transcription in transcription_segments
            ]

            mock_encoded_segments = [
                "VIEW NODE 0".encode(),
                "NODE 0 MOVE UP 50".encode(),
            ]

            Log(f"Encoded Segments: {encoded_segments}")
            Log(f"Encoded Mock Segments: {mock_encoded_segments}")

            if len(encoded_segments) > 0:
                try:
                    com_server.SendMultipartWithMeta(encoded_segments, metadata)
                except:
                    Log("Error Sending Encoded Transcription Segments", LogLevel.ERROR)
            else:
                Log("No Transcription Segments Returned", LogLevel.WARNING)


if __name__ == "__main__":
    main()
```

5.5 Audio.py

```
import gc
```

```python
from itertools import chain as iter_chain
from multiprocessing import Lock

import numpy as np
import av


try:
    from .Logging import Log
except ImportError:
    from Logging import Log


class AudioResampler:
    """Audio Resampler for Resampling Incoming Audio to the Target Sample Rate. Using FFmpeg for Resampling."""

    def __init__(self, target_sample_rate: int = 16000):
        """Constructor of Audio Resampler Class

        Args:
            target_sample_rate (int, optional): The Target for all incoming resampling requests. Defaults to 16000 (Required by Whisper).
        """

        self._audio_resampler = av.AudioResampler(
            format="s16", layout="mono", rate=target_sample_rate
        )
        self._resample_mutex = Lock()

    def __del__(self):
        """Destructor of Audio Resampler Class.

        Ensures PyAV Resampler Object is Properly Deleted, calling Garbage Collection in the process.
        """

        # Try Deleting the resampler object to cleanly free up memory
        try:
            del self._audio_resampler
        except:
            pass

        try:  # Delete the mutex
            del self._resample_mutex
        except:
            pass

        # Force Garbage Collection, due to resampler not being properly deleted otherwise.
        gc.collect()

    def resample(
        self,
        audio_data: np.ndarray,
        buffer_sample_rate: int = 48000,
        buffer_num_channels: int = 2,
    ) -> np.ndarray:
        """Resamples the Incoming Audio Data to the Classes Assigned Target Sample Rate.

        Args:
            audio_data (np.ndarray): Audio Data to Resample.
            buffer_sample_rate (int, optional): Sample Rate of the Incoming Audio Data. Defaults to 48000.
            buffer_num_channels (int, optional): Number of Channels in the Incoming Audio Data. Defaults to 2 (Stereo).

        Returns:
            np.ndarray: Resampled Version of the Incoming Audio Data.
        """

        audio_data = self._convert_to_s16(audio_data).reshape(-1, 1)

        frame: av.AudioFrame = av.AudioFrame.from_ndarray(
            audio_data.T,
            format="s16",
            layout="stereo" if buffer_num_channels == 2 else "mono",
        )

        frame.sample_rate = buffer_sample_rate

        resampled_frames: list[av.AudioFrame] = []
        with self._resample_mutex:
            resampled_frames = self._audio_resampler.resample(frame)

        return self._convert_to_float32(resampled_frames[0].to_ndarray()).reshape(
            -1,
        )

    def _resample_frame(self, frame: av.AudioFrame) -> list[av.AudioFrame]:
        """Resamples an AudioFrame to the target sample rate.

        Args:
            frame (av.AudioFrame): An AudioFrame to resample.

        Returns:
            list[av.AudioFrame]: A List of Resampled AudioFrames generated from the input frame,
        """
        with self._resample_mutex:
            return self._audio_resampler.resample(frame)

    def _resample_frames(self, frames: list[av.AudioFrame]):
        """Resamples an array of AudioFrames to the target sample rate.

        Args:
            frames (list[av.AudioFrame]): An array of AudioFrames to resample.

        Yields:
            An Array of Generators for the Resampled AudioFrames from the frame inputs.
        """

        for frame in iter_chain(frames, [None]):
            yield from self._audio_resampler.resample(frame)

    def _convert_to_float32(self, audio_data: np.ndarray) -> np.ndarray:
        """Converts the provided audio data to float32 format.

        Args:
            audio_data (np.ndarray): The audio data to convert.

        Raises:
            ValueError: If the data type is not supported.

        Returns:
            np.ndarray: The Input Audio Data in float32 format.
        """

        if audio_data.dtype == np.float32:
            return audio_data

        elif audio_data.dtype == np.int16:
            return audio_data.astype(np.float32) / 32768.0

        else:
            raise ValueError("Unsupported data type")

    def _convert_to_s16(self, audio_data: np.ndarray) -> np.ndarray:
        """Converts the provided audio data to int16 format.

        Args:
            audio_data (np.ndarray): The audio data to convert.

        Raises:
            ValueError: If the data type is not supported.

        Returns:
            np.ndarray: The Input Audio Data in int16 format.
        """

        if audio_data.dtype == np.int16:
            return audio_data

        elif audio_data.dtype == np.float32:
            return (audio_data * 32768.0).astype(np.int16)

        else:
            raise ValueError("Unsupported data type")
```

5.6 CommunicationServer.py

```python
import numpy as np
import json
import zmq

try:
    from .Logging import Log, LogLevel
except ImportError:
    from Logging import Log, LogLevel


class CommunicationServer:
    """Communication Server Class for Handling Communication Between Python and C++.

    Using ZeroMQ for Socket Communication. (Push / PULL Architecture)
    """

    def __init__(
        self,
        send_socket_type: int,
        recv_socket_type: int,
        send_socket_addr: str = "tcp://127.0.0.1:5556",
        recv_socket_addr: str = "tcp://127.0.0.1:5555",
        poll_timeout: int = 10,
    ):
        """Constructor of Communication Server Class

        Args:
            send_socket_type (int): ZeroMQ Socket Type for Sending Messages.
            recv_socket_type (int): ZeroMQ Socket Type for Receiving Messages.
            send_socket_addr (str, optional): Local Address / Port for Sending Communication Data. Defaults to "tcp://127.0.0.1:5556".
            recv_socket_addr (str, optional): Local Address / Port for Receiving Communication Data. Defaults to "tcp://127.0.0.1:5555".
            poll_timeout (int, optional): Amount of time (ms) for event polling on the Receive Socket. Defaults to 10.
        """

        # Create the Context
        self.context = zmq.Context()

        # Create a Socket
        self.send_socket: zmq.Socket = self.context.socket(send_socket_type)
        self.send_socket_context = self.send_socket.connect(send_socket_addr)

        self.recv_socket = self.context.socket(recv_socket_type)
        self.recv_socket_context = self.recv_socket.bind(recv_socket_addr)

        self.poller = zmq.Poller()
        self.poller.register(self.recv_socket, zmq.POLLIN)
        self.poller_timeout_time = poll_timeout

    def __del__(self):
        """Destructor of Communication Server Class.

        Closes the Sockets and Terminates the ZeroMQ Context.
        """

        self.send_socket.close()
        self.recv_socket.close()

        self.context.term()

    def EventOccured(self) -> bool:
        """Checks if a Receive Event has Occured on the Receive Socket.

        Returns:
            bool: True if an Event has Occured, False Otherwise.
        """

        polled_events = dict(self.poller.poll(self.poller_timeout_time))
        if len(polled_events) > 0 and polled_events.get(self.recv_socket) == zmq.POLLIN:
            return True
        else:
            return False

    def SendString(self, message: str) -> bool:
        """Sends a String Message on the Send Socket.

        Args:
            message (str): String Message to Send.

        Returns:
            bool: True if the Message was Sent Successfully, False Otherwise.
        """

        try:
            self.send_socket.send_string(message)
            return True
        except:
            Log("CommunicationServer | Error Sending String Message", LogLevel.WARNING)
            return False

    def SendJSON(self, message: dict) -> bool:
        """Sends a JSON Message on the Send Socket.

        Args:
            message (dict): Stringified JSON Message to Send.

        Returns:
            bool: True if the Message was Sent Successfully, False Otherwise.
        """

        try:
            self.send_socket.send_json(message)
            return True
        except:
            Log(
                "CommunicationServer | Error Sending JSON Message",
                LogLevel.WARNING,
            )
            return False

    def SendNDArray(self, message: np.ndarray) -> bool:
        """Sends a Numpy NDArray Message on the Send Socket.

        Args:
            message (np.ndarray): NDArray of Data to Send.

        Returns:
            bool: True if the Data was Sent Successfully, False Otherwise.
        """

        try:
            self.send_socket.send(message)
            return True
        except:
            Log(
                "CommunicationServer | Error Sending NDArray Message",
                LogLevel.WARNING,
            )
            return False

    def SendNDArrayWithMeta(self, message: np.ndarray, meta: dict) -> bool:
        """Sends a Numpy NDArray Message with Metadata on the Send Socket.

        Args:
            message (np.ndarray): NDArray of Data to Send.
            meta (dict): A Dictionary of Metadata to Send.

        Returns:
            bool: True if the Data was Sent Successfully, False Otherwise.
        """

        try:
            self.send_socket.send_multipart([json.dumps(meta).encode(), message.data])

            return True
        except:
            Log(
                "CommunicationServer | Error Sending NDArray With Meta Message",
                LogLevel.WARNING,
            )
            return False

    def SendMultipart(self, message: list) -> bool:
        """Sends a Multipart Message on the Send Socket.

        Args:
            message (list): List of Messages to Send.

        Returns:
            bool: True if the MultiPart Message was Sent Successfully, False Otherwise.
        """

        try:
            self.send_socket.send_multipart(message)
            return True
        except:
            Log(
                "CommunicationServer | Error Sending Multipart Message",
                LogLevel.WARNING,
            )
            return False

    def SendMultipartWithMeta(self, message: list, meta: dict) -> bool:
        """Sends a Multipart Message with Metadata on the Send Socket.

        Args:
            message (list): List of Messages to Send.
            meta (dict): Metadata to Send.

        Returns:
            bool: True if the MultiPart Message with Metadata was Sent Successfully, False Otherwise.
        """

        try:
            self.send_socket.send_multipart([json.dumps(meta).encode(), *message])
            return True
        except:
            Log(
                "CommunicationServer | Error Sending Multipart With Meta Message",
                LogLevel.WARNING,
            )
            return False

    def RecieveRaw(self):
        """Receives a Raw Message of Bytes from the Receive Socket.

        Returns:
            bytes: Raw Received Bytes from the Receive Socket.
        """

        return self.recv_socket.recv(zmq.DONTWAIT)

    def ReceiveString(self) -> str:
        """Receives a String Message from the Receive Socket.

        Returns:
            str: Received String Message.
        """

        return self.recv_socket.recv_string(zmq.DONTWAIT)

    def ReceiveJSON(self):
        """Receive a JSON Message from the Receive Socket.

        Returns:
            dict: Dictionary of the Received JSON Message.
        """

        return json.loads(self.recv_socket.recv_json(zmq.DONTWAIT))

    def ReceiveNDArray(self, dtype=np.float32) -> np.ndarray:
        """Receives a Numpy NDArray from the Receive Socket.

        Args:
            dtype (optional): Type of NDArray of Received Data. Defaults to np.float32.

        Returns:
            np.ndarray: Receieved NDArray Message.
        """

        return np.frombuffer(
            self.recv_socket.recv(zmq.DONTWAIT),
            dtype=dtype,
        )

    def ReceiveNDArrayWithMeta(self, dtype=np.float32) -> tuple[np.ndarray, dict]:
        """Receives a Numpy NDArray with Metadata from the Receive Socket.

        Args:
            dtype (optional): Type of NDArray of Received Data. Defaults to np.float32.

        Returns:
            tuple[np.ndarray, dict]: Tuple of Received NDArray and Dict Metadata Object.
        """

        recv_message = self.recv_socket.recv_multipart(zmq.DONTWAIT)

        if len(recv_message) > 1:
            return (
                np.frombuffer(recv_message[1], dtype=dtype),
                json.loads(recv_message[0]),
            )

        elif len(recv_message) == 1:
            Log(
                "CommunicationServer | Error Receiving NDArray With Meta. Only Contains One Message,
Assumed Data.",
                LogLevel.WARNING,
            )
            return (np.frombuffer(recv_message[0], dtype=dtype), {})

        Log("CommunicationServer | Error Receiving NDArray With Meta", LogLevel.WARNING)

    def ReceiveMultipart(self) -> list[bytes]:
        """Receieved a Raw Multipart Message from the Receive Socket.

        Returns:
            list[bytes]: Raw List of Bytes from the Received Multipart Message.
        """

        return self.recv_socket.recv_multipart(zmq.DONTWAIT)
```

5.7 LibUtils.py

```python
import sys
import os


def append_paths_to_library_path(paths: list[str]):
    """Appends the given paths to the systems active library path.

    Args:
        paths (list[str]): List of Paths to Append.
    """

    sys.path.extend(paths)


def get_path_list() -> list[str]:
    """Gets a list of paths in the PATH environment variable.

    Returns:
        list[str]: _description_
    """

    return os.getenv("PATH").split(";")


def get_filtered_path_list(filter_list: list[str]) -> list[str]:
    """Gets a list of paths in the PATH environment variable that contain any of the given filters.

    Args:
        filter_list (list[str]): List of Filter Strings to Search for in the PATH env.

    Returns:
        list[str]: List of Found Paths.
    """

    return [
        path for path in get_path_list() for filter in filter_list if filter in path
    ]


def get_child_directories(path: str, depth: int = 0) -> list[str]:
    """
    Recursively searches the given directory, for any further child directories.

    Args:
        path (str): The path to the directory to search.
        depth (int): The depth to search for child directories. Defaults to 0.
    """

    assert os.path.isdir(path), f"Path: {path} is not a directory."

    return [
        root
        for root, _, _ in os.walk(path, topdown=True)
        if root[len(path) :].count(os.sep) < depth
    ]
```

5.8 Logging.py

```python
from enum import Enum


class LogLevel(Enum):
    INFO = 0
    WARNING = 1
    ERROR = 2


def Log(message: str, log_level: LogLevel = LogLevel.INFO):
    """Logs a Message to the Unreal Engine Console.

    Displays the given message in the Unreal Engine Console, with the given log level.
    If the Unreal Engine Python API is not available, the message is printed to the python terminal.

    Args:
        message (str): Message to Log.
        log_level (LogLevel, optional): Log Level of the Message. Defaults to LogLevel.INFO.
    """

    message = f"|| LogOpenAccessibilityPy || {message} ||"

    try:
        from unreal import log, log_warning, log_error

        if log_level == LogLevel.INFO:
            log(message)
        elif log_level == LogLevel.WARNING:
            log_warning(message)
        elif log_level == LogLevel.ERROR:
            log_error(message)
        else:
            log(message)

    except ImportError:
        print(message)
        pass
```

5.9 WhisperInterface.py

```
from ctypes import Union
import numpy as np

from faster_whisper import WhisperModel
from faster_whisper.transcribe import Segment

try:
    from .Logging import Log, LogLevel
except ImportError:
    from Logging import Log, LogLevel


class WhisperInterface:
    """Interface Class for Interacting with the CTranslate2 Faster Whisper Model."""

    def __init__(
        self,
        model_name: str = "distil-small.en",
        device: str = "auto",
        cpu_threads: int = 4,
        transcribe_workers: int = 2,
        compute_type: str = "default",
    ):
        """Constructor of Whisper Interface Class

        Args:
            model_name (str, optional): Hugging-Face Model Specifier for CTranslate Compatible Models.
 Defaults to "distil-small.en".
            device (str, optional): Device host for the Whisper Model (Can be "auto", "cpu", "cuda").
 Defaults to "auto".
            cpu_threads (int, optional): Amount of CPU Threads to use, if Hosting the Model on a CPU.
 Defaults to 4.
            transcribe_workers (int, optional): Amount of Thread Workers for Audio Transcription.
 Defaults to 2.
            compute_type (str, optional): Data Structure for Whisper Compute. Defaults to "default".
        """

        # Whisper Focused Variables
        self.whisper_model = WhisperModel(
            model_name,
            device=device,
            compute_type=compute_type,
            num_workers=transcribe_workers,
            cpu_threads=cpu_threads,
            local_files_only=True,
        )
        self.beam_size = 5

    def __del__(self):
        """Destructor of Whisper Interface Class."""

        del self.whisper_model

    def process_file_from_dir(self, filepath: str):
        """Transcribes an Audio Files from a Given WAV File Path.

        Args:
            filepath (str): Path to the Audio Files to Transcribe.

        Returns:
            A List of Segments containing the Transcribed Text and their Time Stamps.
        """

        segments, info = self.whisper_model.transcribe(
            filepath,
            beam_size=self.beam_size,
            language="en",
            prepend_punctuations="",
            append_punctuations="",
            vad_filter=True,
        )

        Log(
            f"WhisperInterface | Detected Language: {info.language} | Probability:
{info.language_probability} | Duration: {info.duration}"
        )

        for segment in segments:
            Log(
                f"WhisperInterface | Segment : {segment.text} | Start: {segment.start} | End:
{segment.end}"
            )

        return list(segments)

    def process_audio_buffer(
        self, audio_buffer: np.ndarray
    ) -> tuple[list[Segment], dict]:
        """Transcribes an NDArray AudioBuffer.

        Args:
            audio_buffer (np.ndarray): AudioBuffer to Transcribe.

        Returns:
            tuple[list[Segment], dict]: Tuple Containing a List of Transcription Segments and a
Dictionary of Collected Metadata.
        """

        segments, info = self.whisper_model.transcribe(
            audio_buffer,
            beam_size=self.beam_size,
            language="en",
        )

        Log(
            f"WhisperInterface || Detected Language: {info.language} | Probability:
{info.language_probability} | Duration: {info.duration}"
        )

        collected_metadata = {
            "duration": info.duration,
            "language": info.language,
            "language_probability": info.language_probability,
        }

        return list(segments), collected_metadata
```

5.10 TestWhisper.py

```
import numpy as np
from faster_whisper import WhisperModel
from faster_whisper.audio import decode_audio
import time


class ModelInfo:
    name: str
    time_to_load: float
    time_to_transcribe: float
    time_total: float


def test_whisper_model(model_name: str, audiobuffer) -> ModelInfo:

    model_info = ModelInfo()
    model_info.name = model_name

    # --------------------------------
    # Model Initialization
    # --------------------------------

    start_time = time.perf_counter()

    whisper_model = WhisperModel(model_name, device="cuda", compute_type="default")

    end_time = time.perf_counter()

    model_info.time_to_load = end_time - start_time

    # ---------------------------------
    # Audio Transcription
    # ---------------------------------

    start_time = time.perf_counter()

    segments, _ = whisper_model.transcribe(audiobuffer, beam_size=5)

    end_time = time.perf_counter()

    model_info.time_to_transcribe = end_time - start_time

    model_info.time_total = model_info.time_to_load + model_info.time_to_transcribe

    # ---------------------------------
    # Show Segments
    # ---------------------------------

    for segment in segments:
        print(
            f"|| WhisperInterface || Start: {segment.start} | End: {segment.end} | Text: {segment.text} ||"
        )

    # ---------------------------------

    print(
        f"\n{model_info.name}:\nTime To Load: {model_info.time_to_load}\nTime To Transcribe: {model_info.time_to_transcribe}\nTotal Time: {model_info.time_total}\n"
    )

    return model_info


# ---------------------------------
# Testing Here
# ---------------------------------

filepath = "D:\dev\Unreal Engine\AccessibilityProject\Plugins\OpenAccessibility\Saved\BouncedWavFiles\OpenAccessibility\Audioclips\Captured_User_

models_to_test = ["tiny", "base", "small", "Systran/faster-distil-whisper-small.en"]

audiobuffer = decode_audio(filepath)

input_help = "\n"
for index, model in enumerate(models_to_test):
    input_help += f"{index}: {model}\n"

user_input = input(f"Models: {input_help}\nor Leave Empty to Test All:\n").lower()

if user_input == "":

    print(f"Testing All Models")

    for model in models_to_test:
        info = test_whisper_model(model, audiobuffer)

        print(
            f"Model: {info.name} | Time To Load: {info.time_to_load} | Time To Transcribe: {info.time_to_transcribe} | Total Time: {info.time_total}"
        )

else:
    if models_to_test.__contains__(user_input):
        test_whisper_model(user_input, audiobuffer)

    elif user_input.isdigit() and int(user_input) < len(models_to_test):
        test_whisper_model(models_to_test[int(user_input)], audiobuffer)
```

5.11 OpenAccessibility.Build.cs

```
// Copyright Epic Games, Inc. All Rights Reserved.

using System.IO;
using UnrealBuildTool;

public class OpenAccessibility : ModuleRules
{
    public OpenAccessibility(ReadOnlyTargetRules Target) : base(Target)
    {
        PCHUsage = ModuleRules.PCHUsageMode.UseExplicitOrSharedPCHs;

        PublicIncludePaths.AddRange(
            new string[] {
                // ... add public include paths required here ...
            }
            );

        PrivateIncludePaths.AddRange(
            new string[] {
                // ... add other private include paths required here ...
            }
            );


        PublicDependencyModuleNames.AddRange(
            new string[]
            {
                "Core",
                // ... add other public dependencies that you statically link with here ...
            }
            );


        PrivateDependencyModuleNames.AddRange(
            new string[]
            {
                // Internal Plugin Modules
                "OpenAccessibilityCommunication",

                // Core Modules
                "CoreUObject",
                "Engine",

                // Editor Modules
                "UnrealEd",
                "GraphEditor",
                "Kismet",
                "AIModule",

                // Slate UI
                "Slate",
                "SlateCore",
                "EditorStyle",
            }
            );


        DynamicallyLoadedModuleNames.AddRange(
            new string[]
            {
                // ... add any modules that your module loads dynamically here ...
            }
            );

        CircularlyReferencedDependentModules.AddRange(
            new string[]
            {

            }
        );
    }
}
```

5.12 SAccessibilityTranscriptionVis.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "AccessibilityWidgets/SAccessibilityTranscriptionVis.h"

SAccessibilityTranscriptionVis::~SAccessibilityTranscriptionVis()
{
}

void SAccessibilityTranscriptionVis::Construct(const FArguments& InArgs)
{
    // Transcription Holder
    TSharedPtr<SVerticalBox> TranscriptionHolder = SNew(SVerticalBox)
        + SVerticalBox::Slot()
        .Padding(4.0f)
        .AutoHeight();

    // Verify a least one slot will be constructed
    int TranscriptionSlotAmount = FMath::Max(1, InArgs._VisAmount);

    FSlateFontInfo FontInfo = FAppStyle::GetFontStyle("NormalText");
    FontInfo.Size = 12;

    TSharedPtr<STextBlock> CurrentTranscriptionSlot;
    for (int i = 0; i < TranscriptionSlotAmount; i++)
    {
        TranscriptionHolder->AddSlot()
            .HAlign(HAlign_Center)
            .Padding(4.0f)
            .AutoHeight()
            [
                SAssignNew(CurrentTranscriptionSlot, STextBlock)
                    .Text(FText::GetEmpty())
                    .Font(FontInfo)
                    .SimpleTextMode(true)
                    .ColorAndOpacity(i == 0 ? FSlateColor(FLinearColor(1.0f, 1.0f, 0, 1.0f)) : FSlateColor(FLinearColor(0.5f, 0.5f, 0.5f, 1.0f)))
            ];

        TranscriptionSlots.Add(CurrentTranscriptionSlot);
    }

    // Construct the Main Component

    ChildSlot
    .Padding(FMargin(5.0f))
    [
        SNew(SOverlay)
        + SOverlay::Slot()
        .ZOrder(1)
        [
            SNew(SBorder)
            .BorderBackgroundColor(FSlateColor(FLinearColor::Gray))
            [
                SNew(SBox)
                .MinDesiredWidth(250.0f)
                .MinDesiredHeight(60.0f)
                [
                    TranscriptionHolder.ToSharedRef()
                ]
            ]
        ]
    ];

    this->TranscriptionContainer = TranscriptionHolder;
}

void SAccessibilityTranscriptionVis::Tick(const FGeometry& AllottedGeometry, const double InCurrentTime, const float InDeltaTime)
{
    SBox::Tick(AllottedGeometry, InCurrentTime, InDeltaTime);
}

void SAccessibilityTranscriptionVis::UpdateTopTranscription(const FString& InTopTranscription)
{
    FString LastTopText = InTopTranscription;
    FString TempText;

    TSharedPtr<STextBlock> CurrentTranscriptionSlot;
    for (TWeakPtr<STextBlock>& TranscriptionSlot : TranscriptionSlots)
    {
        CurrentTranscriptionSlot = TranscriptionSlot.Pin();

        TempText = FString(CurrentTranscriptionSlot->GetText().ToString());
        CurrentTranscriptionSlot->SetText(FText::FromString(LastTopText));

        CurrentTranscriptionSlot->Invalidate(EInvalidateWidgetReason::PaintAndVolatility);

        LastTopText = TempText;
    }

    TranscriptionContainer.Pin()->Invalidate(EInvalidateWidget::Layout);
}
```

5.13 SContentIndexer.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "AccessibilityWidgets/SContentIndexer.h"
#include "AccessibilityWidgets/SIndexer.h"

SContentIndexer::~SContentIndexer()
{
}

void SContentIndexer::Construct(const FArguments& InArgs)
{
    TSharedPtr<SWidget> Content;

    switch (InArgs._IndexPositionToContent)
    {
        case EIndexerPosition::Top:
            Content = ConstructTopIndexer(InArgs);
            break;

        case EIndexerPosition::Bottom:
            Content = ConstructBottomIndexer(InArgs);
            break;

        default:
        case EIndexerPosition::Left:
            Content = ConstructLeftIndexer(InArgs);
            break;

        case EIndexerPosition::Right:
            Content = ConstructRightIndexer(InArgs);
            break;
    }

    ChildSlot
    [
        Content.ToSharedRef()
    ];
}

void SContentIndexer::Tick(const FGeometry& AllottedGeometry, const double InCurrentTime, const float
 InDeltaTime)
{
    SBox::Tick(AllottedGeometry, InCurrentTime, InDeltaTime);
}

void SContentIndexer::UpdateIndex(const int32 IndexValue)
{
    if (IndexerWidget.IsValid())
        IndexerWidget.Pin()->UpdateIndex( IndexValue );
}

TSharedPtr<SWidget> SContentIndexer::ConstructTopIndexer(const FArguments& InArgs)
{
    return SNew(SVerticalBox)
    .Visibility(AccessWidgetVisibilityAttribute(InArgs._ContentToIndex.ToSharedRef()))

        + SVerticalBox::Slot()
        .HAlign(HAlign_Center)
        .VAlign(VAlign_Center)
        .AutoHeight()
        .Padding(.1f, .25f)
        [
            ConstructIndexContainer(InArgs).ToSharedRef()
        ]

        + SVerticalBox::Slot()
        .HAlign(HAlign_Center)
        .VAlign(VAlign_Center)
        .AutoHeight()
        [
            ConstructContentContainer(InArgs._ContentToIndex.ToSharedRef()).ToSharedRef()
        ];
}

TSharedPtr<SWidget> SContentIndexer::ConstructBottomIndexer(const FArguments& InArgs)
{
    return SNew(SVerticalBox)
    .Visibility(AccessWidgetVisibilityAttribute(InArgs._ContentToIndex.ToSharedRef()))

        + SVerticalBox::Slot()
        .HAlign(HAlign_Center)
        .VAlign(VAlign_Center)
        .AutoHeight()
        [
            ConstructContentContainer(InArgs._ContentToIndex.ToSharedRef()).ToSharedRef()
        ]

        + SVerticalBox::Slot()
        .HAlign(HAlign_Center)
        .VAlign(VAlign_Center)
        .AutoHeight()
        .Padding(.1f, .25f)
        [
            ConstructIndexContainer(InArgs).ToSharedRef()
        ];
}

TSharedPtr<SWidget> SContentIndexer::ConstructLeftIndexer(const FArguments& InArgs)
{
    return SNew(SHorizontalBox)
    .Visibility(AccessWidgetVisibilityAttribute(InArgs._ContentToIndex.ToSharedRef()))

        + SHorizontalBox::Slot()
        .VAlign(VAlign_Center)
        .HAlign(HAlign_Center)
        .AutoWidth()
        .Padding(.25f, .1f)
        [
            ConstructIndexContainer(InArgs).ToSharedRef()
        ]

        + SHorizontalBox::Slot()
        .VAlign(VAlign_Center)
        .HAlign(HAlign_Center)
        .AutoWidth()
        [
            ConstructContentContainer(InArgs._ContentToIndex.ToSharedRef()).ToSharedRef()
        ];
}

TSharedPtr<SWidget> SContentIndexer::ConstructRightIndexer(const FArguments& InArgs)
{
    return SNew(SHorizontalBox)
    .Visibility(AccessWidgetVisibilityAttribute(InArgs._ContentToIndex.ToSharedRef()))

        + SHorizontalBox::Slot()
        .VAlign(VAlign_Center)
        .HAlign(HAlign_Center)
        .AutoWidth()
        [
            ConstructContentContainer(InArgs._ContentToIndex.ToSharedRef()).ToSharedRef()
        ]

        + SHorizontalBox::Slot()
        .VAlign(VAlign_Center)
        .HAlign(HAlign_Center)
        .AutoWidth()
        .Padding(.25f, .1f)
        [
            ConstructIndexContainer(InArgs).ToSharedRef()
        ];
}

TSharedPtr<SWidget> SContentIndexer::ConstructContentContainer(TSharedRef<SWidget> ContentToIndex)
{
    IndexedContent = ContentToIndex;
    return ContentToIndex;
}

TSharedPtr<SWidget> SContentIndexer::ConstructIndexContainer(const FArguments& InArgs, FLinearColor
  TextColor)
{
    return SAssignNew(IndexerWidget, SIndexer)
    .TextColor(TextColor)
    .BorderColor(FLinearColor::Gray)
    .IndexValue(InArgs._IndexValue)
    .IndexVisibility(InArgs._IndexVisibility);
}

FText SContentIndexer::ConstructIndexText(int32 Index)
{
    return FText::FromString(FString::FromInt(Index));
}
```

5.14 SIndexer.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "AccessibilityWidgets/SIndexer.h"

SIndexer::~SIndexer()
{

}

void SIndexer::Tick(const FGeometry& AllotedGeometry, const double InCurrentTime,  const float
  InDeltaTime)
```

```
{
    SBox::Tick(AllotedGeometry, InCurrentTime, InDeltaTime);
}

void SIndexer::Construct(const FArguments& InArgs)
{
    ChildSlot
    [
        SNew(SBorder)
        .HAlign(HAlign_Center)
        .VAlign(VAlign_Center)
        .Visibility(InArgs._IndexVisibility)
        .Padding(FMargin(4.f, 2.f))
        .BorderBackgroundColor( FSlateColor(InArgs._BorderColor) )
        [
            SAssignNew(IndexTextBlock, STextBlock)
            .Text( FText::FromString(FString::FromInt(InArgs._IndexValue)) )
            .TextShapingMethod( ETextShapingMethod::KerningOnly )
            .ColorAndOpacity( FSlateColor(InArgs._TextColor) )
        ]
    ];
}

void SIndexer::UpdateIndex(const int32 NewIndex)
{
    if (!IndexTextBlock.IsValid())
        return;

    IndexTextBlock.Pin()->SetText(
        FText::FromString( FString::FromInt(NewIndex) )
    );
}

void SIndexer::UpdateIndex(const FString& NewIndex)
{
    if (!IndexTextBlock.IsValid())
        return;

    IndexTextBlock.Pin()->SetText(
        FText::FromString(NewIndex)
    );
}

void SIndexer::UpdateIndex(const FText& NewIndex)
{
    if (!IndexTextBlock.IsValid())
        return;

    IndexTextBlock.Pin()->SetText(NewIndex);
}
```

5.15 AccessibilityAddNodeContextMenu.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "AccessibilityWrappers/AccessibilityAddNodeContextMenu.h"
#include "OpenAccessibilityLogging.h"

#include "Widgets/Input/SSearchBox.h"

#include "AccessibilityWidgets/SContentIndexer.h"


#include "Styling/AppStyle.h"

UAccessibilityAddNodeContextMenu::UAccessibilityAddNodeContextMenu()
    : UPhraseTreeContextMenuObject()
{

}

UAccessibilityAddNodeContextMenu::UAccessibilityAddNodeContextMenu(TSharedRef<IMenu> Menu)
    : UPhraseTreeContextMenuObject(Menu)
{

}

UAccessibilityAddNodeContextMenu::UAccessibilityAddNodeContextMenu(TSharedRef<IMenu> Menu,
 TSharedRef<SGraphActionMenu> GraphMenu)
: UPhraseTreeContextMenuObject(Menu)
{
    this->GraphMenu = GraphMenu;
    this->FilterTextBox = GraphMenu->GetFilterTextBox();
```

```cpp
}

UAccessibilityAddNodeContextMenu::UAccessibilityAddNodeContextMenu(TSharedRef<IMenu> Menu, TSharedRef<SGraphActionMenu> GraphMenu, TSharedRef<STreeView<TSharedPtr<FGraphActionNode»> TreeView)
: UPhraseTreeContextMenuObject(Menu)
{
    this->GraphMenu = GraphMenu;
    this->TreeView = TreeView;
    this->FilterTextBox = GraphMenu->GetFilterTextBox();
}

UAccessibilityAddNodeContextMenu::~UAccessibilityAddNodeContextMenu()
{

}

void UAccessibilityAddNodeContextMenu::Init(TSharedRef<IMenu> InMenu, TSharedRef<FPhraseNode> InContextRoot)
{
    Init(InMenu);

    this->ContextRoot = InContextRoot;
}

void UAccessibilityAddNodeContextMenu::Init(TSharedRef<IMenu> InMenu)
{
    UPhraseTreeContextMenuObject::Init(InMenu);

    // This is a Mess but half the Menu Containers are private, so have to move myself to key aspects of the Menu.

    TSharedPtr<SWidget> KeyboardFocusedWidget = StaticCastSharedPtr<SEditableText>(
        FSlateApplication::Get().GetKeyboardFocusedWidget()
    );
    if (!KeyboardFocusedWidget.IsValid())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("AddNodeContextWrapper::Init: KeyboardFocusedWidget is Invalid."));
        return;
    }

    this->GraphMenu = StaticCastSharedPtr<SGraphActionMenu>(
        KeyboardFocusedWidget
        ->GetParentWidget()
        ->GetParentWidget()
        ->GetParentWidget()
        ->GetParentWidget()
        ->GetParentWidget()
    );

    {
        TSharedPtr<SSearchBox> SearchBox = StaticCastSharedPtr<SSearchBox>(
            KeyboardFocusedWidget
                ->GetParentWidget()
                ->GetParentWidget()
                ->GetParentWidget()
        );

        TSharedRef<SWidget> SearchBoxSibling = SearchBox->GetParentWidget()->GetChildren()->GetChildAt(1);
        this->TreeView = StaticCastSharedRef<STreeView<TSharedPtr<FGraphActionNode»>(
            SearchBoxSibling->GetChildren()->GetChildAt(0)->GetChildren()->GetChildAt(0)
        );
    }

    {
        TSharedRef<SCheckBox> CheckBox = StaticCastSharedRef<SCheckBox>(
this->GraphMenu.Pin()->GetParentWidget()->GetChildren()->GetChildAt(0)->GetChildren()->GetChildAt(2)
        );

        this->ContextAwarenessCheckBox = CheckBox;
    }

    this->FilterTextBox = this->GraphMenu.Pin()->GetFilterTextBox();

    FSlateApplication::Get().SetKeyboardFocus(this->TreeView.Pin());
}

void UAccessibilityAddNodeContextMenu::Init(TSharedRef<IMenu> InMenu, TSharedRef<SGraphActionMenu> InGraphMenu, TSharedRef<STreeView<TSharedPtr<FGraphActionNode»> InTreeView)
{
    UPhraseTreeContextMenuObject::Init(InMenu);

    this->GraphMenu = InGraphMenu;
    this->TreeView = InTreeView;
    this->FilterTextBox = InGraphMenu->GetFilterTextBox();
```

```
}

bool UAccessibilityAddNodeContextMenu::Tick(float DeltaTime)
{
    if (!GraphMenu.IsValid() || !Menu.IsValid())
        return false;

    if (DoesItemsRequireRefresh())
        RefreshAccessibilityWidgets();

    TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

    // Set Previous Vars For Next Tick
    PrevFilterString = FilterTextBox.Pin()->GetText().ToString();
    PrevNumItemsBeingObserved = TreeViewPtr->GetNumItemsBeingObserved();
    PrevNumGeneratedChildren = TreeViewPtr->GetNumGeneratedChildren();
    PrevScrollDistance = TreeViewPtr->GetScrollDistance().Y;

    return true;
}

bool UAccessibilityAddNodeContextMenu::Close()
{
    RemoveTickDelegate();
    Menu.Pin()->Dismiss();

    return true;
}

void UAccessibilityAddNodeContextMenu::ScaleMenu(const float ScaleFactor)
{
    // Scale TreeView Element
    {
        TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

        TreeViewPtr->SetItemHeight(16 * ScaleFactor);
    }

    // Scale Window Element
    {
        TSharedPtr<SWindow> WindowPtr = Window.Pin();

        WindowPtr->SetSizingRule(ESizingRule::UserSized);
        WindowPtr->Resize(WindowPtr->GetSizeInScreen() * ScaleFactor);
    }
}

bool UAccessibilityAddNodeContextMenu::DoesItemsRequireRefresh()
{
    TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

    return (
        FilterTextBox.Pin()->GetText().ToString() != PrevFilterString ||
        TreeViewPtr->GetNumItemsBeingObserved() != PrevNumItemsBeingObserved ||
        TreeViewPtr->GetNumGeneratedChildren() != PrevNumGeneratedChildren ||
        TreeViewPtr->GetScrollDistance().Y != PrevScrollDistance
    );
}

void UAccessibilityAddNodeContextMenu::RefreshAccessibilityWidgets()
{

    TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

    TArray<TSharedPtr<FGraphActionNode» Items =
 TArray<TSharedPtr<FGraphActionNode»(TreeViewPtr->GetRootItems());

    {
        TSharedPtr<STableRow<TSharedPtr<FGraphActionNode»> ItemWidget = nullptr;

        while (Items.Num() > 0)
        {
            const TSharedPtr<FGraphActionNode> Item = Items[0];
            Items.RemoveAt(0);

            if (TreeViewPtr->IsItemExpanded(Item))
                Items.Append(Item->Children);

            ItemWidget = StaticCastSharedPtr<STableRow<TSharedPtr<FGraphActionNode»>(
                TreeViewPtr->WidgetFromItem(Item)
            );

            if (!ItemWidget.IsValid())
                continue;

            // TO-DO: Change To Non-HardCoded Type Comparison.
            if (ItemWidget->GetContent()->GetType() == "SContentIndexer")
            {
                UpdateAccessibilityWidget(ItemWidget.ToSharedRef());
            }
            else
            {
                ApplyAccessibilityWidget(ItemWidget.ToSharedRef());
            }
        }
    }
}

FGraphActionNode* UAccessibilityAddNodeContextMenu::GetGraphActionFromIndex(const int32 InIndex)
{
    TArrayView<const TSharedPtr<FGraphActionNode» Items = TreeView.Pin()->GetItems();

    if (Items.Num() > InIndex)
        return Items[InIndex].Get();

    else return nullptr;
}

void UAccessibilityAddNodeContextMenu::GetGraphActionFromIndex(const int32 InIndex, FGraphActionNode*
 OutGraphAction)
{
    TArrayView<const TSharedPtr<FGraphActionNode» Items = TreeView.Pin()->GetItems();

    if (Items.Num() > InIndex)
        OutGraphAction = Items[InIndex].Get();

    else OutGraphAction = nullptr;
}

TSharedPtr<FGraphActionNode> UAccessibilityAddNodeContextMenu::GetGraphActionFromIndexSP(const int32
 InIndex)
{
    if (TreeView.Pin()->GetItems().Num() <= InIndex)
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("GetGraphActionFromIndexSP: Provided Index is Out
 of Range."));
        return nullptr;
    }
    return TreeView.Pin()->GetItems()[InIndex];
}

void UAccessibilityAddNodeContextMenu::SelectGraphAction(const int32 InIndex)
{
    TSharedPtr<FGraphActionNode> GraphAction = GetGraphActionFromIndexSP(InIndex);

    if (GraphAction.IsValid())
    {
        TreeView.Pin()->Private_OnItemClicked(GraphAction);
    }
    else
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("SelectGraphAction: Provided GraphAction is
 Invalid."));
    }
}

void UAccessibilityAddNodeContextMenu::PerformGraphAction(const int32 InIndex)
{
    TSharedPtr<FGraphActionNode> GraphAction = GetGraphActionFromIndexSP(InIndex);

    if (!GraphAction.IsValid())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("PerformGraphAction: Provided GraphAction is
 Invalid."));
    }

    if (GraphAction->IsActionNode())
    {
        TreeView.Pin()->Private_ClearSelection();
        TreeView.Pin()->Private_SetItemSelection(GraphAction, true, true);
        TreeView.Pin()->Private_SignalSelectionChanged(ESelectInfo::OnMouseClick);
    }
    else
    {
        TreeView.Pin()->Private_OnItemDoubleClicked(GraphAction);
    }
}

FString UAccessibilityAddNodeContextMenu::GetFilterText()
{
    return FilterTextBox.Pin()->GetText().ToString();
}

void UAccessibilityAddNodeContextMenu::SetFilterText(const FString& InFilterText)
```

```
{
    FilterTextBox.Pin()->SetText(FText::FromString(InFilterText));
}

void UAccessibilityAddNodeContextMenu::AppendFilterText(const FString& InFilterText)
{
    FilterTextBox.Pin()->SetText(
        FText::FromString(
            FilterTextBox.Pin()->GetText().ToString() + TEXT(" ") + InFilterText
        )
    );
}

void UAccessibilityAddNodeContextMenu::ResetFilterText()
{
    FilterTextBox.Pin()->SetText(FText::FromString(TEXT("")));
}

void UAccessibilityAddNodeContextMenu::SetScrollDistance(const float InScrollDistance)
{
    TreeView.Pin()->SetScrollOffset(InScrollDistance);
}

void UAccessibilityAddNodeContextMenu::AppendScrollDistance(const float InScrollDistance)
{
    if (TreeView.Pin()->GetScrollOffset() + InScrollDistance < 0.0f)
    {
        TreeView.Pin()->SetScrollOffset(0.0f);
        return;
    }

    TreeView.Pin()->AddScrollOffset(InScrollDistance, true);
}

void UAccessibilityAddNodeContextMenu::SetScrollDistanceTop()
{
    TreeView.Pin()->ScrollToTop();
}

void UAccessibilityAddNodeContextMenu::SetScrollDistanceBottom()
{
    TreeView.Pin()->ScrollToBottom();
}

void UAccessibilityAddNodeContextMenu::ToggleContextAwareness()
{
    ContextAwarenessCheckBox.Pin()->ToggleCheckedState();
}

void
 UAccessibilityAddNodeContextMenu::ApplyAccessibilityWidget(TSharedRef<STableRow<TSharedPtr<FGraphActionNode»>
 ItemWidget)
{
    TSharedPtr<SWidget> ItemContent = ItemWidget->GetContent();

    ItemWidget->SetContent(
        SNew(SContentIndexer)
        .IndexValue(ItemWidget->GetIndexInList())
        .IndexPositionToContent(EIndexerPosition::Left)
        .ContentToIndex(ItemContent)
    );
}

void
 UAccessibilityAddNodeContextMenu::UpdateAccessibilityWidget(TSharedRef<STableRow<TSharedPtr<FGraphActionNode»>
 ItemWidget)
{
    TSharedPtr<SContentIndexer> ItemContent =
 StaticCastSharedPtr<SContentIndexer>(ItemWidget->GetContent());

    ItemContent->UpdateIndex(ItemWidget->GetIndexInList());
}
```

5.16 AccessibilityGraphEditorContext.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "AccessibilityWrappers/AccessibilityGraphEditorContext.h"

#include "OpenAccessibilityLogging.h"
#include "AccessibilityWidgets/SIndexer.h"
#include "AccessibilityWidgets/SContentIndexer.h"
#include "Utils/WidgetUtils.h"
```

```

#include "Widgets/SWindow.h"
#include "Widgets/Input/SEditableTextBox.h"

UAccessibilityGraphEditorContext::UAccessibilityGraphEditorContext()
    : Super()
{

}

void UAccessibilityGraphEditorContext::Init(TSharedRef<IMenu> InMenu, TSharedRef<FPhraseNode>
 InContextRoot)
{
    Super::Init(InMenu, InContextRoot);

    TSharedRef<SWindow> WindowRef = Window.Pin().ToSharedRef();

    if (!FindGraphActionMenu(WindowRef))
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphEditorContext: Cannot Find a SGraphActionMenu
 Widget"));
    }

    if (!FindStaticComponents(WindowRef))
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphEditorContext: Cannot Find Any Static
 Components"));
    }

    if (!FindTreeView(WindowRef))
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphEditorContext: Cannot Find a STreeView
 Widget"));
    }
    else
    {
        TreeViewTickRequirements = FTreeViewTickRequirements();
    }
}

bool UAccessibilityGraphEditorContext::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

    if (TreeViewCanTick())
    {
        TickTreeViewAccessibility();

        TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

        TreeViewTickRequirements.PrevSearchText = FilterTextBox.Pin()->GetText().ToString();
        TreeViewTickRequirements.PrevNumGeneratedChildren = TreeViewPtr->GetNumGeneratedChildren();
        TreeViewTickRequirements.PrevNumItemsBeingObserved = TreeViewPtr->GetNumItemsBeingObserved();
        TreeViewTickRequirements.PrevScrollDistance = TreeViewPtr->GetScrollDistance().Y;
    }

    return true;
}

bool UAccessibilityGraphEditorContext::Close()
{
    Super::Close();

    return true;
}

void UAccessibilityGraphEditorContext::ScaleMenu(const float ScaleFactor)
{
    Super::ScaleMenu(ScaleFactor);

    // Scale TreeView
    if (TreeView.IsValid())
    {
        TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

        TreeViewPtr->SetItemHeight(16 * ScaleFactor);
    }

    // Scale Window Element
    if (Window.IsValid())
    {
        TSharedPtr<SWindow> WindowPtr = Window.Pin();

        WindowPtr->SetSizingRule(ESizingRule::UserSized);
        WindowPtr->Resize(WindowPtr->GetSizeInScreen() * ScaleFactor);
    }
}
```

```

TSharedPtr<FGraphActionNode> UAccessibilityGraphEditorContext::GetTreeViewAction(const int32& InIndex)
{
    TArrayView<const TSharedPtr<FGraphActionNode>> Items = TreeView.Pin()->GetItems();

    if (TreeView.IsValid() && Items.Num() > InIndex && InIndex >= 0)
        return TreeView.Pin()->GetItems()[InIndex];

    return TSharedPtr<FGraphActionNode>();
}

void UAccessibilityGraphEditorContext::SelectAction(const int32& InIndex)
{
    if (InIndex < 0)
        return;

    if (!CheckBoxes.IsEmpty() && InIndex < CheckBoxes.Num())
    {
        if (CheckBoxes[InIndex].IsValid())
        {
            CheckBoxes[InIndex].Pin()->ToggleCheckedState();
            return;
        }
    }

    TSharedPtr<FGraphActionNode> ChosenTreeViewAction = GetTreeViewAction(InIndex -
 GetStaticIndexOffset());
    if (!ChosenTreeViewAction.IsValid())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("SelectGraphAction: Provided TreeView Action is
 Invalid"))
        return;
    }

    auto TreeViewPtr = TreeView.Pin();
    if (ChosenTreeViewAction->IsActionNode())
    {
        TreeViewPtr->Private_ClearSelection();
        TreeViewPtr->Private_SetItemSelection(ChosenTreeViewAction, true, true);
        TreeViewPtr->Private_SignalSelectionChanged(ESelectInfo::Type::OnMouseClick);
    }
    else
    {
        TreeViewPtr->Private_OnItemDoubleClicked(ChosenTreeViewAction);
    }
}

FString UAccessibilityGraphEditorContext::GetFilterText()
{
    return FilterTextBox.IsValid() ? FilterTextBox.Pin()->GetText().ToString() : FString();
}

void UAccessibilityGraphEditorContext::SetFilterText(const FString& NewString)
{
    if (!FilterTextBox.IsValid())
        return;

    FilterTextBox.Pin()->SetText(
        FText::FromString(NewString)
    );
}

void UAccessibilityGraphEditorContext::AppendFilterText(const FString& StringToAdd)
{
    if (!FilterTextBox.IsValid())
        return;

    TSharedPtr<SEditableTextBox> FilterTextBoxPtr = FilterTextBox.Pin();

    FilterTextBoxPtr->SetText(
        FText::FromString( FilterTextBoxPtr->GetText().ToString() + TEXT(" ") + StringToAdd )
    );
}

void UAccessibilityGraphEditorContext::SetScrollDistance(const float NewDistance)
{
    if (TreeView.IsValid())
        return;

    TreeView.Pin()->SetScrollOffset(NewDistance);
}

void UAccessibilityGraphEditorContext::AppendScrollDistance(const float DistanceToAdd)
{
    auto TreeViewPtr = TreeView.Pin();

    if (TreeViewPtr->GetScrollOffset() + DistanceToAdd < 0.0f)
    {
        TreeViewPtr->SetScrollOffset(0.0f);
        return;
    }

    TreeViewPtr->AddScrollOffset(DistanceToAdd);
}

void UAccessibilityGraphEditorContext::SetScrollDistanceTop()
{
    TreeView.Pin()->ScrollToTop();
}

void UAccessibilityGraphEditorContext::SetScrollDistanceBottom()
{
    TreeView.Pin()->ScrollToBottom();
}

const int32 UAccessibilityGraphEditorContext::GetStaticIndexOffset()
{
    return CheckBoxes.Num();
}

bool UAccessibilityGraphEditorContext::FindGraphActionMenu(const TSharedRef<SWidget>& SearchRoot)
{
    TSharedPtr<SGraphActionMenu> GraphActionMenu = GetWidgetDescendant<SGraphActionMenu>(SearchRoot,
 TEXT("SGraphActionMenu"));
    if (GraphActionMenu.IsValid())
    {
        GraphMenu = GraphActionMenu;
        FilterTextBox = GraphActionMenu->GetFilterTextBox();

        return true;
    }

    return false;
}

bool UAccessibilityGraphEditorContext::FindTreeView(const TSharedRef<SWidget>& SearchRoot)
{
    TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> ContextTreeView =
 GetWidgetDescendant<STreeView<TSharedPtr<FGraphActionNode»>(
        SearchRoot,
        TEXT("STreeView<TSharedPtr<FGraphActionNode»")
    );
    if (ContextTreeView.IsValid())
    {
        TreeView = ContextTreeView;

        return true;
    }

    return false;
}

bool UAccessibilityGraphEditorContext::FindStaticComponents(const TSharedRef<SWidget>& SearchRoot)
{
    TArray<FSlotBase*> FoundComponentSlots = GetWidgetSlotsByType(
        SearchRoot,
        TSet<FString> {
            TEXT("SCheckBox")
        }
    );

    if (!FoundComponentSlots.IsEmpty())
    {
        // Sort and Index the Static Components.
        for (int i = 0; i < FoundComponentSlots.Num(); i++)
        {
            FSlotBase* FoundComponentSlot = FoundComponentSlots[i];

            TSharedPtr<SWidget> DetachedWidget = FoundComponentSlot->DetachWidget();
            if (!DetachedWidget.IsValid())
                continue;

            int32 ComponentIndex = -1;
            FString ComponentType = DetachedWidget->GetTypeAsString();

            if (ComponentType == "SCheckBox")
            {
                ComponentIndex = CheckBoxes.Num();
                CheckBoxes.Add(StaticCastSharedPtr<SCheckBox>(DetachedWidget));
            }

            FoundComponentSlot->AttachWidget(
                SNew(SContentIndexer)
                .IndexValue(ComponentIndex)
                .IndexPositionToContent(EIndexerPosition::Left)
                .ContentToIndex(DetachedWidget)
            );
        }

        return true;
    }

    return false;
}

bool UAccessibilityGraphEditorContext::TreeViewCanTick()
{
    return TreeView.IsValid() && GraphMenu.IsValid();
}

bool UAccessibilityGraphEditorContext::TreeViewRequiresTick()
{
    if (!TreeView.IsValid() || !GraphMenu.IsValid())
        return false;

    bool bFilterTextChange = FilterTextBox.IsValid()
        ? FilterTextBox.Pin()->GetText().ToString() != TreeViewTickRequirements.PrevSearchText
        : false;

    TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

    return (
        bFilterTextChange ||
        TreeViewPtr->GetNumItemsBeingObserved() != TreeViewTickRequirements.PrevNumItemsBeingObserved
 ||
        TreeViewPtr->GetNumGeneratedChildren() != TreeViewTickRequirements.PrevNumGeneratedChildren ||
        TreeViewPtr->GetScrollDistance().Y != TreeViewTickRequirements.PrevScrollDistance
    );
}

void UAccessibilityGraphEditorContext::TickTreeViewAccessibility()
{
    if (!TreeViewRequiresTick())
        return;

    TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeViewPtr = TreeView.Pin();

    TArray<TSharedPtr<FGraphActionNode» Items = TArray<TSharedPtr<FGraphActionNode»(
        TreeViewPtr->GetRootItems()
    );


    TSharedPtr<STableRow<TSharedPtr<FGraphActionNode»> ItemWidget = nullptr;
    const int32 IndexOffset = GetStaticIndexOffset();

    while (Items.Num() > 0)
    {
        const TSharedPtr<FGraphActionNode> Item = Items[0];
        Items.RemoveAt(0);

        if (TreeViewPtr->IsItemExpanded(Item))
            Items.Append(Item->Children);

        ItemWidget = StaticCastSharedPtr<STableRow<TSharedPtr<FGraphActionNode»>(
            TreeViewPtr->WidgetFromItem(Item)
        );
        if (!ItemWidget.IsValid())
            continue;

        TSharedPtr<SWidget> ItemContent = ItemWidget->GetContent();

        if (ItemContent->GetType() == "SContentIndexer")
        {
            UpdateAccessibilityWidget(
                StaticCastSharedRef<SContentIndexer>(ItemContent.ToSharedRef()),
                IndexOffset + ItemWidget->GetIndexInList()
            );
        }
        else
        {
            ItemWidget->SetContent(
                CreateAccessibilityWrapper(ItemContent.ToSharedRef(), IndexOffset +
 ItemWidget->GetIndexInList())
            );
        }
    }
}

void UAccessibilityGraphEditorContext::UpdateAccessibilityWidget(const TSharedRef<SContentIndexer>&
 ContentIndexer, const int32& NewIndex)
{
    ContentIndexer->UpdateIndex(NewIndex);
}

const TSharedRef<SContentIndexer> UAccessibilityGraphEditorContext::CreateAccessibilityWrapper(const TSharedRef<SWidget>& ContentToWrap, const int32& Index)
{
    return SNew(SContentIndexer)
        .IndexValue(Index)
        .IndexPositionToContent(EIndexerPosition::Left)
        .ContentToIndex(ContentToWrap);
}

```

5.17 AccessibilityGraphLocomotionContext.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "AccessibilityWrappers/AccessibilityGraphLocomotionContext.h"
#include "AccessibilityWidgets/SIndexer.h"
#include "OpenAccessibilityLogging.h"

#include "SGraphPanel.h"

UAccessibilityGraphLocomotionContext::UAccessibilityGraphLocomotionContext(const FObjectInitializer& ObjectInitializer)
    : UPhraseTreeContextObject()
{
    LinkedEditor = TWeakPtr<SGraphEditor>();
}

UAccessibilityGraphLocomotionContext::~UAccessibilityGraphLocomotionContext()
{
    Close();

    UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: CONTEXT DESTROYED."));
}

void UAccessibilityGraphLocomotionContext::Init()
{
    {
        TSharedPtr<SDockTab> ActiveTab = FGlobalTabmanager::Get()->GetActiveTab();
        if (!ActiveTab.IsValid())
        {
            UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: NO ACTIVE TAB FOUND."));
            return;
        }

        LinkedEditor = StaticCastSharedRef<SGraphEditor>(ActiveTab->GetContent());
        if (!LinkedEditor.IsValid())
        {
            UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: CURRENT ACTIVE TAB IS NOT OF TYPE - SGraphEditor"));
            return;
        }
    }

    TSharedPtr<SGraphEditor> LinkedEditorPtr = LinkedEditor.Pin();

    Init(LinkedEditorPtr.ToSharedRef());
}

void UAccessibilityGraphLocomotionContext::Init(const TSharedRef<SGraphEditor> InGraphEditor)
{
    LinkedEditor = InGraphEditor;

    InGraphEditor->GetViewLocation(StartViewPosition, StartViewZoom);
    InGraphEditor->ZoomToFit(false);

    CreateVisualGrid(InGraphEditor);
    GenerateVisualChunks(InGraphEditor, FIntVector2(6, 4));

    HideNativeVisuals();

    BindFocusChangedEvent();
}

bool UAccessibilityGraphLocomotionContext::SelectChunk(const int32& Index)
{
    if (Index > ChunkArray.Num() || Index < 0)
        return false;

    const FGraphLocomotionChunk SelectedChunk = ChunkArray[Index];

    const SGraphPanel* LinkedPanel = LinkedEditor.Pin()->GetGraphPanel();

    const FVector2D GraphTopLeftCoord =
 LinkedPanel->PanelCoordToGraphCoord(SelectedChunk.GetChunkTopLeft());
    const FVector2D GraphBottomRightCoord =
 LinkedPanel->PanelCoordToGraphCoord(SelectedChunk.GetChunkBottomRight());

    ChangeChunkVis(Index, FLinearColor::Red);

    GEditor->GetTimerManager()->SetTimer(
        SelectionTimerHandle,
        [this, Index, GraphTopLeftCoord, GraphBottomRightCoord]()
        {
            ChangeChunkVis(Index);

            if (MoveViewport(GraphTopLeftCoord, GraphBottomRightCoord))
            {
                if (CurrentViewPosition != FVector2D::ZeroVector)
                    PreviousPositions.Push(CurrentViewPosition);

                CurrentViewPosition = FPanelViewPosition(GraphTopLeftCoord, GraphBottomRightCoord);
            }
            else
            {
                UE_LOG(LogOpenAccessibility, Log, TEXT("Failed To Jump To Viewport Coords (TopLeft: %s
 | BottomRight: %s)"),
                    *GraphTopLeftCoord.ToString(), *GraphBottomRightCoord.ToString());
            }
        },
        0.5f,
        false
    );

    return true;
}

bool UAccessibilityGraphLocomotionContext::RevertToPreviousView()
{
    if (PreviousPositions.IsEmpty())
    {
        LinkedEditor.Pin()->ZoomToFit(false);
        return true;
    }

    if (!MoveViewport(PreviousPositions.Pop()))
    {
        return false;
    }

    return true;
}

void UAccessibilityGraphLocomotionContext::ConfirmSelection()
{
    Close();
}

void UAccessibilityGraphLocomotionContext::CancelLocomotion()
{
    if (LinkedEditor.IsValid())
    {
        LinkedEditor.Pin()->SetViewLocation(StartViewPosition, StartViewZoom);

        Close();
    }
}

bool UAccessibilityGraphLocomotionContext::Close()
{
    UnbindFocusChangedEvent();

    if (SelectionTimerHandle.IsValid())
        GEditor->GetTimerManager()->ClearTimer(SelectionTimerHandle);

    RemoveVisualGrid();
    UnHideNativeVisuals();

    bIsActive = false;

    RemoveFromRoot();
    MarkAsGarbage();

    UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: CONTEXT CLOSED."));

    return true;
}

```

```
bool UAccessibilityGraphLocomotionContext::MoveViewport(const FVector2D& InTopLeft, const FVector2D& InBottomRight) const
{
    if (!LinkedEditor.IsValid())
        return false;

    TSharedPtr<SGraphEditor> LinkedEditorPtr = LinkedEditor.Pin();
    SGraphPanel* LinkedPanel = LinkedEditorPtr->GetGraphPanel();

    return LinkedPanel->JumpToRect(InTopLeft, InBottomRight);
}

bool UAccessibilityGraphLocomotionContext::MoveViewport(const FPanelViewPosition& NewViewPosition) const
{
    if (!LinkedEditor.IsValid())
        return false;

    SGraphPanel* LinkedPanel = LinkedEditor.Pin()->GetGraphPanel();

    return LinkedPanel->JumpToRect(NewViewPosition.TopLeft, NewViewPosition.BotRight);
}

void UAccessibilityGraphLocomotionContext::ChangeChunkVis(const int32& Index, const FLinearColor& NewColor)
{
    check(Index < ChunkArray.Num() && Index >= 0)

    ChunkArray[Index].SetVisColor(NewColor);
}

void UAccessibilityGraphLocomotionContext::CreateVisualGrid(const TSharedRef<SGraphEditor> InGraphEditor)
{
    TSharedPtr<SOverlay> GraphViewport = StaticCastSharedPtr<SOverlay>(InGraphEditor->GetGraphPanel()->GetParentWidget());
    if (!GraphViewport.IsValid())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: NO GRAPH VIEWPORT FOUND."));
        return;
    }

    GridParent = GraphViewport;

    GraphViewport->AddSlot()
    .ZOrder(1)
    .VAlign(VAlign_Fill)
    .HAlign(HAlign_Fill)
    [
        SAssignNew(GridContainer, SUniformGridPanel)
    ];
}

void UAccessibilityGraphLocomotionContext::GenerateVisualChunks(const TSharedRef<SGraphEditor> InGraphEditor, FIntVector2 InVisualChunkSize)
{
    ChunkArray.Reset(InVisualChunkSize.X * InVisualChunkSize.Y);
    ChunkSize = InVisualChunkSize;

    TSharedPtr<SUniformGridPanel> GridContainerPtr = GridContainer.Pin();

    int32 ChunkIndex = -1;
    TSharedPtr<SBox> ChunkWidget;
    TSharedPtr<SBorder> ChunkVisWidget;
    TSharedPtr<SIndexer> ChunkIndexer;

    for (int32 Y = 0; Y < InVisualChunkSize.Y; Y++)
    {
        for (int32 X = 0; X < InVisualChunkSize.X; X++)
        {
            ChunkIndex = X + (Y * InVisualChunkSize.X);
            FGraphLocomotionChunk& GraphChunk = ChunkArray.EmplaceAt_GetRef(ChunkIndex);

            GridContainerPtr->AddSlot(X, Y)
            [
                SAssignNew(ChunkWidget, SBox)
                [
                    SAssignNew(ChunkVisWidget, SBorder)
                    .Padding(0.5f)
                    .BorderBackgroundColor(FLinearColor::Yellow)
                    [
                        SNew(SBorder)
                        .HAlign(HAlign_Center)
                        .VAlign(VAlign_Center)
                        .BorderBackgroundColor(FLinearColor::Yellow)
                        [
                            SAssignNew(ChunkIndexer, SIndexer)
```

```
                                .TextColor(FLinearColor::Yellow)
                                .IndexValue(ChunkIndex)
                            ]
                        ]
                    ]
                ];

                GraphChunk.ChunkWidget = ChunkWidget;
                GraphChunk.ChunkVisWidget = ChunkVisWidget;
                GraphChunk.ChunkIndexer = ChunkIndexer;
            }
        }

        CalculateVisualChunksBounds();
}

void UAccessibilityGraphLocomotionContext::CalculateVisualChunksBounds()
{
    if (!LinkedEditor.IsValid())
        return;

    SGraphPanel* LinkedPanel = LinkedEditor.Pin()->GetGraphPanel();
    FVector2D PanelGeoSize = LinkedPanel->GetTickSpaceGeometry().GetLocalSize();

    double ChunkWidgetSizeX = PanelGeoSize.X / ChunkSize.X;
    double ChunkWidgetSizeY = PanelGeoSize.Y / ChunkSize.Y;

    FGraphLocomotionChunk Chunk;
    double ChunkX, ChunkY;

    int32 ArrIndex;
    for (int Y = 0; Y < ChunkSize.Y; Y++)
    {
        for (int X = 0; X < ChunkSize.X; X++)
        {
            ArrIndex = (Y * ChunkSize.X) + X;

            Chunk = ChunkArray[ArrIndex];

            ChunkX = X * ChunkWidgetSizeX;
            ChunkY = Y * ChunkWidgetSizeY;

            Chunk.SetChunkBounds(
                FVector2D(ChunkX, ChunkY),
                FVector2D(ChunkWidgetSizeX + ChunkX, ChunkWidgetSizeY + ChunkY)
            );

            ChunkArray[ArrIndex] = Chunk;
        }
    }
}

void UAccessibilityGraphLocomotionContext::RemoveVisualGrid()
{
    TSharedPtr<SUniformGridPanel> GridContainerPtr = GridContainer.Pin();
    if (GridContainerPtr.IsValid())
    {
        TSharedPtr<SOverlay> ParentWidget = StaticCastSharedPtr<SOverlay>(
            GridContainerPtr->GetParentWidget()
        );

        if (ParentWidget.IsValid()) {
            ParentWidget->RemoveSlot(GridContainerPtr.ToSharedRef());

            GridParent = ParentWidget;
        }
        else UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: PARENT WIDGET NOT FOUND,
 CANNOT REMOVE LOCOMOTION WIDGETS."))
    }
}

void UAccessibilityGraphLocomotionContext::HideNativeVisuals()
{
    NativeWidgetVisibility.Empty();

    TSharedPtr<SOverlay> GraphViewport = GridParent.Pin();
    TSharedPtr<SUniformGridPanel> VisualGrid = GridContainer.Pin();
    SGraphPanel* GraphPanel = LinkedEditor.Pin()->GetGraphPanel();

    FChildren* ViewportChildren = GraphViewport->GetChildren();

    TSharedPtr<SWidget> ChildWidget;
    for (int32 i = 0; i < ViewportChildren->Num(); i++)
    {
        ChildWidget = ViewportChildren->GetChildAt(i);

        if (ChildWidget != VisualGrid && ChildWidget.Get() != GraphPanel)
        {
            NativeWidgetVisibility.Add(ChildWidget.Get(), ChildWidget->GetVisibility());

            ChildWidget->SetVisibility(EVisibility::Hidden);
        }
    }
}

void UAccessibilityGraphLocomotionContext::UnHideNativeVisuals()
{
    FChildren* ViewportChildren = GridParent.Pin()->GetChildren();

    TSharedPtr<SWidget> ChildWidget;
    for (int32 i = 0; i < ViewportChildren->Num(); i++)
    {
        ChildWidget = ViewportChildren->GetChildAt(i);

        if (NativeWidgetVisibility.Contains(ChildWidget.Get()))
        {
            ChildWidget->SetVisibility(NativeWidgetVisibility[ChildWidget.Get()]);
        }
    }

    NativeWidgetVisibility.Empty();
}

void UAccessibilityGraphLocomotionContext::OnFocusChanged(
    const FFocusEvent& FocusEvent,
    const FWeakWidgetPath& OldFocusedWidgetPath, const TSharedPtr<SWidget>& OldFocusedWidget,
    const FWidgetPath& NewFocusedWidgetPath, const TSharedPtr<SWidget>& NewFocusedWidget
)
{
    if (!bIsActive)
        return;

    UE_LOG(LogOpenAccessibility, Warning, TEXT("GraphLocomotion: FOCUS CHANGED."));

    TSharedPtr<SGraphEditor> LinkedEditorPtr = LinkedEditor.Pin();

    if (!NewFocusedWidgetPath.ContainsWidget(LinkedEditorPtr.ToSharedRef()))
    {
        bIsActive = false;
        Close();
    }
}

void UAccessibilityGraphLocomotionContext::BindFocusChangedEvent()
{
    FocusChangedHandle = FSlateApplication::Get().OnFocusChanging()
        .AddUObject(this, &UAccessibilityGraphLocomotionContext::OnFocusChanged);
}

void UAccessibilityGraphLocomotionContext::UnbindFocusChangedEvent()
{
    if (FocusChangedHandle.IsValid())
    {
        FSlateApplication::Get().OnFocusChanging().Remove(FocusChangedHandle);
    }
}
```

5.18 AccessibilityWindowToolbar.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "AccessibilityWrappers/AccessibilityWindowToolbar.h"
#include "AccessibilityWidgets/SContentIndexer.h"

#include "PhraseTree/Containers/ParseRecord.h"
#include "PhraseTree/Containers/Input/UParseIntInput.h"

UAccessibilityWindowToolbar::UAccessibilityWindowToolbar() : UObject()
{
    LastToolkit = TWeakPtr<SWidget>();
    LastTopWindow = TWeakPtr<SWindow>();
    LastToolkitParent = TWeakPtr<SBorder>();

    ConsoleCommands.Add(IConsoleManager::Get().RegisterConsoleCommand(
        TEXT("OpenAccessibiliy.ToolBar.ShowIndexerStats"),
        TEXT("Displays the Indexer Stats for the Toolbar."),

        FConsoleCommandDelegate::CreateLambda([this]() {
            UE_LOG(LogOpenAccessibility, Display, TEXT("| ToolBar Indexer Stats | Indexed Amount: %d |
"), ToolbarIndex.Num())
```

```
        })
    ));

    BindTicker();
}

UAccessibilityWindowToolbar::~UAccessibilityWindowToolbar()
{
    UE_LOG(LogOpenAccessibility, Log, TEXT("AccessibilityToolBar: Destroyed."));

    UnbindTicker();
}

bool UAccessibilityWindowToolbar::Tick(float DeltaTime)
{
    TSharedPtr<SWindow> TopWindow = FSlateApplication::Get().GetActiveTopLevelRegularWindow();
    if (!TopWindow.IsValid())
    {
        return true;
    }

    TSharedPtr<SBorder> ContentContainer;
    if (TopWindow != LastTopWindow)
        ContentContainer = GetWindowContentContainer(TopWindow.ToSharedRef());
    else ContentContainer = LastToolkitParent.Pin();

    if (!ContentContainer.IsValid())
    {
        return true;
    }


    TSharedPtr<SWidget> Toolkit = ContentContainer->GetContent();
    if (!Toolkit.IsValid())
    {
        return true;
    }

    if (ApplyToolbarIndexing(Toolkit.ToSharedRef(), TopWindow.ToSharedRef()))
    {
        LastToolkit = Toolkit;
        //UE_LOG(LogOpenAccessibility, Log, TEXT("AccessibilityToolBar: Toolkit Indexing Applied To
 %s"), *Toolkit->GetTypeAsString());
    }

    LastTopWindow = TopWindow;
    LastToolkitParent = ContentContainer;

    return true;
}

bool UAccessibilityWindowToolbar::ApplyToolbarIndexing(TSharedRef<SWidget> ToolkitWidget,
 TSharedRef<SWindow> ToolkitWindow)
{
    TSharedPtr<SWidget> ToolBarContainer;
    if (!GetToolKitToolBar(ToolkitWidget, ToolBarContainer))
    {
        UE_LOG(LogOpenAccessibility, Log, TEXT("Failed to get Toolbar."));
        return false;
    }

    if (!ToolBarContainer.IsValid())
    {
        UE_LOG(LogOpenAccessibility, Log, TEXT("Toolbar Container Is Not Valid."));
        return false;
    }

    TArray<FChildren*> ChildrenToFilter = TArray<FChildren*> {
        ToolBarContainer->GetChildren()
    };

    FString WidgetType;
    TSet<FString> AllowedWidgetTypes = TSet<FString>{
        TEXT("SToolBarButtonBlock"),
        TEXT("SToolBarComboButtonBlock"),
        TEXT("SToolBarStackButtonBlock"),
        TEXT("SUniformToolBarButtonBlock")
    };

    ToolbarIndex.Reset();

    int32 Index = -1;
    while (ChildrenToFilter.Num() > 0)
    {
        FChildren* Children = ChildrenToFilter[0];
        ChildrenToFilter.RemoveAt(0);

        // To-Do: Learn How to Write Readable Code.
        for (int i = 0; i < Children->NumSlot(); i++)
        {
            FSlotBase& ChildSlot = const_cast<FSlotBase&>(Children->GetSlotAt(i));

            TSharedPtr<SWidget> ChildWidget = Children->GetChildAt(i);
            if (!ChildWidget.IsValid() || ChildWidget->GetDesiredSize() == FVector2D::ZeroVector)
                continue;

            WidgetType = ChildWidget->GetTypeAsString();

            if (ChildWidget.IsValid() && AllowedWidgetTypes.Contains(WidgetType))
            {
                TSharedPtr<SMultiBlockBaseWidget> ToolBarButtonWidget =
       StaticCastSharedPtr<SMultiBlockBaseWidget>(ChildWidget);

                ChildSlot.DetachWidget();

                ToolbarIndex.GetKeyOrAddValue(
                    ToolBarButtonWidget.Get(),
                    Index
                );

                ChildSlot.AttachWidget(
                    SNew(SContentIndexer)
                    .IndexValue(Index)
                    .IndexPositionToContent(EIndexerPosition::Bottom)
                    .ContentToIndex(ToolBarButtonWidget)
                    .IndexVisibility_Lambda([this, ToolkitWidget]() -> EVisibility {
                            return (this->IsActiveToolbar(ToolkitWidget))
                            ? EVisibility::Visible
                            : EVisibility::Hidden;
                    })
                );
            }
            else if (ChildWidget.IsValid() && WidgetType == "SContentIndexer")
            {
                TSharedPtr<SContentIndexer> IndexerWidget =
       StaticCastSharedPtr<SContentIndexer>(ChildWidget);

                TSharedPtr<SMultiBlockBaseWidget> IndexedContent =
       StaticCastSharedRef<SMultiBlockBaseWidget>(IndexerWidget->GetContent());
                if (!IndexedContent.IsValid())
                    continue;

                ToolbarIndex.GetKeyOrAddValue(
                    IndexedContent.Get(),
                    Index
                );

                IndexerWidget->UpdateIndex(Index);
            }
            else ChildrenToFilter.Add(ChildWidget->GetChildren());
        }
    }

    return true;
}

// -- Util Widget Function --

template<typename T = SWidget>
FORCEINLINE TSharedPtr<T> GetWidgetDescendantOfType(TSharedRef<SWidget> Widget, FName TypeName)
{
    if (Widget->GetType() == TypeName)
    {
        return Widget;
    }

    TArray<FChildren*> ChildrenToFilter;
    ChildrenToFilter.Add(Widget->GetChildren());

    while (ChildrenToFilter.Num() > 0)
    {
        FChildren* Children = ChildrenToFilter.Pop();

        for (int i = 0; i < Children->Num(); i++)
        {
            TSharedRef<SWidget> Child = Children->GetChildAt(i);

            ChildrenToFilter.Add(Child->GetChildren());

            if (Child->GetType() == TypeName)
            {
                return StaticCastSharedPtr<T>(Child.ToSharedPtr());
            }
        }
    }

    return nullptr;
}

// --  --

void UAccessibilityWindowToolbar::SelectToolbarItem(int32 Index)
{
    if (ToolbarIndex.IsEmpty())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("ToolBar Index is Empty."))
        return;
    }

    SMultiBlockBaseWidget* LinkedButton;
    if (!ToolbarIndex.GetValue(Index, LinkedButton))
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided Index is Not Linked to a ToolBar
 Button."))
        return;
    }

    TSharedPtr<const FMultiBlock> MultiBlock = LinkedButton->GetBlock();
    if (!MultiBlock.IsValid())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided ToolBar MultiBlock is Not Valid."))
    }

    TSharedPtr<const FUICommandList> ActionList = MultiBlock->GetActionList();
    TSharedPtr<const FUICommandInfo> Action = MultiBlock->GetAction();

    if (ActionList.IsValid() && Action.IsValid())
    {
        ActionList->ExecuteAction( Action.ToSharedRef() );
    }
    else
    {
        const FUIAction& DirectAction = MultiBlock->GetDirectActions();

        DirectAction.Execute();
    }
}

bool UAccessibilityWindowToolbar::IsActiveToolbar(const TSharedRef<SWidget>& ToolkitWidget)
{
    return LastToolkit.IsValid()
        ? LastToolkit.Pin() == ToolkitWidget
        : false;
}

TSharedPtr<SWidget> UAccessibilityWindowToolbar::GetActiveToolkitWidget() const
{
    if (LastToolkit.IsValid())
        return LastToolkit.Pin();

    return TSharedPtr<SWidget>();
}

TSharedPtr<SBorder> UAccessibilityWindowToolbar::GetWindowContentContainer(TSharedRef<SWindow>
 WindowToFindContainer)
{
    // Find SDockingTabStack
    TSharedPtr<SWidget> DockingTabStack = GetWidgetDescendantOfType(WindowToFindContainer,
 "SDockingTabStack");
    if (!DockingTabStack.IsValid())
    {
        UE_LOG(LogOpenAccessibility, Log, TEXT("DockingTabStack is not Valid"));
        return nullptr;
    }

    return StaticCastSharedRef<SBorder>(
        DockingTabStack
            ->GetChildren()->GetChildAt(0) // SVerticalBox
            ->GetChildren()->GetChildAt(1) // SOverlay
            ->GetChildren()->GetChildAt(0) // SBorder
    );
}

bool UAccessibilityWindowToolbar::GetToolKitToolBar(TSharedRef<SWidget> ToolKitWidget,
 TSharedPtr<SWidget>& OutToolBar)
{
    TSharedPtr<SWidget> CurrChild;
    FChildren* CurrChildren = ToolKitWidget->GetChildren();
    if (CurrChildren->Num() == 0)
        return false;

    CurrChild = CurrChildren->GetChildAt(0); // Get SVerticalBox
    CurrChildren = CurrChild->GetChildren();
    if (CurrChildren->Num() == 0)
        return false;

    OutToolBar = CurrChildren->GetChildAt(0); // Get SHorizontalBox
    if (!OutToolBar.IsValid())
        return false;

    return true;
}

void UAccessibilityWindowToolbar::BindTicker()
{
    FTickerDelegate TickDelegate = FTickerDelegate::CreateUObject(this, &UAccessibilityWindowToolbar::Tick);

    TickDelegateHandle = FTSTicker::GetCoreTicker()
        .AddTicker(TickDelegate);
}

void UAccessibilityWindowToolbar::UnbindTicker()
{
    FTSTicker::GetCoreTicker()
        .RemoveTicker(TickDelegateHandle);

}
```

5.19 AssetAccessibilityRegistry.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "AssetAccessibilityRegistry.h"
#include "OpenAccessibilityLogging.h"
#include "BehaviorTree/BehaviorTree.h"

#include "Subsystems/AssetEditorSubsystem.h"
#include "EdGraph/EdGraph.h"
#include "EdGraph/EdGraphNode.h"
#include "MaterialGraph/MaterialGraph.h"

#include "UObject/Class.h"
#include "Misc/Guid.h"

FAssetAccessibilityRegistry::FAssetAccessibilityRegistry()
{
    GraphAssetIndex = TMap<FGuid, TSharedPtr<FGraphIndexer»();
    //GameWorldAssetIndex = TMap<FGuid, FGameWorldIndexer*>();

    AssetOpenedInEditorHandle = GEditor->GetEditorSubsystem<UAssetEditorSubsystem>()->OnAssetOpenedInEditor()
        .AddRaw(this, &FAssetAccessibilityRegistry::OnAssetOpenedInEditor);

    AssetEditorRequestCloseHandle = GEditor->GetEditorSubsystem<UAssetEditorSubsystem>()->OnAssetEditorRequestClose()
        .AddRaw(this, &FAssetAccessibilityRegistry::OnAssetEditorRequestClose);
}

FAssetAccessibilityRegistry::~FAssetAccessibilityRegistry()
{
    GEditor->GetEditorSubsystem<UAssetEditorSubsystem>()->OnAssetOpenedInEditor()
        .Remove(AssetOpenedInEditorHandle);

    GEditor->GetEditorSubsystem<UAssetEditorSubsystem>()->OnAssetEditorRequestClose()
        .Remove(AssetEditorRequestCloseHandle);

    EmptyGraphAssetIndex();
}

void FAssetAccessibilityRegistry::OnAssetOpenedInEditor(UObject* OpenedAsset, IAssetEditorInstance* EditorInstance)
{
    UE_LOG(LogOpenAccessibility, Log, TEXT("|| AssetRegistry || Asset { %s } Opened In Editor: { %s } ||"), *OpenedAsset->GetName(), *EditorInstance->GetEditorName().ToString());

    // Find Asset Type for correct Parsing.
    if (UBlueprint* OpenedBlueprint = Cast<UBlueprint>(OpenedAsset))
    {
        UE_LOG(LogOpenAccessibility, Log, TEXT("|| AssetRegistry || Asset { %s } Is A Blueprint ||"), *OpenedBlueprint->GetName());

        RegisterBlueprintAsset(OpenedBlueprint);
    }
    else if (UMaterial* OpenedMaterial = Cast<UMaterial>(OpenedAsset))
    {
        UE_LOG(LogOpenAccessibility, Log, TEXT("|| AssetRegistry || Asset { %s } Is A Material ||"),
       *OpenedMaterial->GetName());

        RegisterMaterialAsset(OpenedMaterial);
    }
    else if (UBehaviorTree* OpenedBehaviorTree = Cast<UBehaviorTree>(OpenedAsset))
    {
        UE_LOG(LogOpenAccessibility, Log, TEXT("|| AssetRegistry || Asset { %s } Is A Behavior Tree
       ||"), *OpenedBehaviorTree->GetName());

        RegisterBehaviorTreeAsset(OpenedBehaviorTree);
    }
}

void FAssetAccessibilityRegistry::OnAssetEditorRequestClose(UObject* ClosingAsset,
       EAssetEditorCloseReason CloseReason)
{
    if (ClosingAsset == nullptr)
        return;

    UE_LOG(LogOpenAccessibility, Log, TEXT("|| AssetRegistry || Asset { %s } Closed | Reason: { %d }
       ||"), *ClosingAsset->GetFName().ToString(), int64(CloseReason));
}

bool FAssetAccessibilityRegistry::IsGraphAssetRegistered(const UEdGraph* InUEdGraph) const
{
    return GraphAssetIndex.Contains(InUEdGraph->GraphGuid);
}

bool FAssetAccessibilityRegistry::RegisterGraphAsset(const UEdGraph* InGraph)
{
    if (!InGraph->IsValidLowLevel())
        return false;

    GraphAssetIndex.Add(InGraph->GraphGuid, MakeShared<FGraphIndexer>(InGraph));

    for (auto& ChildGraph : InGraph->SubGraphs)
    {
        if (!RegisterGraphAsset(ChildGraph))
        {
            UE_LOG(LogOpenAccessibility, Error, TEXT("|| AssetRegistry || Error When Logging Child
       Graph: { %s } From Parent: { %s }||"), *ChildGraph->GetName(), *InGraph->GetName())

            return false;
        }
    }

    return true;
}

bool FAssetAccessibilityRegistry::RegisterGraphAsset(const UEdGraph* InGraph, const
       TSharedRef<FGraphIndexer> InGraphIndexer)
{
    if (!InGraph->IsValidLowLevel())
        return false;

    GraphAssetIndex.Add(InGraph->GraphGuid, InGraphIndexer.ToSharedPtr());

    for (auto& ChildGraph : InGraph->SubGraphs)
    {
        if (!RegisterGraphAsset(ChildGraph))
        {
            UE_LOG(LogOpenAccessibility, Error, TEXT("|| AssetRegistry || Error When Logging Child
       Graph: { %s } From Parent: { %s} ||"), *ChildGraph->GetName(), *InGraph->GetName());
            return false;
        }
    }

    return true;
}

bool FAssetAccessibilityRegistry::UnregisterGraphAsset(const UEdGraph* UEdGraph)
{
    GraphAssetIndex.Remove(UEdGraph->GraphGuid);

    for (auto& ChildGraph : UEdGraph->SubGraphs)
    {
        if (!UnregisterGraphAsset(ChildGraph))
        {
            UE_LOG(LogOpenAccessibility, Error, TEXT("|| AssetRegistry || Error When Unregistering
       Child Graph: { %s } From Parent: { %s }||"), *ChildGraph->GetName(), *UEdGraph->GetName())

            return false;
        }
    }

    return true;
}

void FAssetAccessibilityRegistry::GetAllGraphKeyIndexes(TArray<FGuid>& OutGraphKeys) const
{
    GraphAssetIndex.GetKeys(OutGraphKeys);
}

TArray<FGuid> FAssetAccessibilityRegistry::GetAllGraphKeyIndexes() const
{
    TArray<FGuid> GraphKeys;
    GraphAssetIndex.GetKeys(GraphKeys);

    return GraphKeys;
}

void FAssetAccessibilityRegistry::GetAllGraphIndexes(TArray<TSharedPtr<FGraphIndexer»&
 OutGraphIndexes) const
{
    return GraphAssetIndex.GenerateValueArray(OutGraphIndexes);
}

TArray<TSharedPtr<FGraphIndexer» FAssetAccessibilityRegistry::GetAllGraphIndexes()
{
    TArray<TSharedPtr<FGraphIndexer» GraphIndexArray;

    GraphAssetIndex.GenerateValueArray(GraphIndexArray);

    return GraphIndexArray;
}

bool FAssetAccessibilityRegistry::IsGameWorldAssetRegistered(const UWorld* UWorld) const
{
    throw std::exception("The method or operation is not implemented.");
}

bool FAssetAccessibilityRegistry::RegisterGameWorldAsset(const UWorld* UWorld)
{
    throw std::exception("The method or operation is not implemented.");
}

bool FAssetAccessibilityRegistry::UnregisterGameWorldAsset(const UWorld* UWorld)
{
    throw std::exception("The method or operation is not implemented.");
}

void FAssetAccessibilityRegistry::EmptyGraphAssetIndex()
{
    for (auto& GraphIndexer : GraphAssetIndex)
    {
        GraphIndexer.Value.Reset();
    }

    GraphAssetIndex.Empty();
}

void FAssetAccessibilityRegistry::EmptyGameWorldAssetIndex()
{
    throw std::exception("The method or operation is not implemented.");
}

void FAssetAccessibilityRegistry::RegisterBlueprintAsset(const UBlueprint* InBlueprint)
{
    // Register the Blueprint's Graphs
    TArray<UEdGraph*> Graphs;

    InBlueprint->GetAllGraphs(Graphs);
    for (auto& Graph : Graphs)
    {
        RegisterGraphAsset(Graph);
    }

    // Register the Blueprint's World
    // Some Blueprints have no connected World / GameObjects,
    // so we need to check if the World is valid

    UWorld* BlueprintDebugWorld = InBlueprint->GetWorldBeingDebugged();
    if (BlueprintDebugWorld != nullptr)
    {
        RegisterUWorldAsset(BlueprintDebugWorld);
    }
}

void FAssetAccessibilityRegistry::RegisterMaterialAsset(const UMaterial* InMaterial)
{
    if (InMaterial->MaterialGraph.IsNull())
```

```
        return;

    TSharedPtr<FGraphIndexer> GraphIndexer =
 MakeShared<FGraphIndexer>(InMaterial->MaterialGraph.Get());

    RegisterGraphAsset(InMaterial->MaterialGraph.Get(), GraphIndexer.ToSharedRef());
}

void FAssetAccessibilityRegistry::RegisterBehaviorTreeAsset(const UBehaviorTree* InBehaviorTree)
{
    if (InBehaviorTree->BTGraph->IsValidLowLevel())
    {
        RegisterGraphAsset(InBehaviorTree->BTGraph);
    }
}

void FAssetAccessibilityRegistry::RegisterUWorldAsset(const UWorld* InWorld)
{
    throw std::exception("The method or operation is not implemented.");
}
```

5.20 GraphIndexer.cpp

```
// Copyright F-Dudley. All Rights Reserved.


#include "GraphIndexer.h"

#include "EdGraph/EdGraph.h"
#include "EdGraph/EdGraphNode.h"
#include "EdGraph/EdGraphPin.h"
#include "GraphEditAction.h"
#include "OpenAccessibilityLogging.h"

FGraphIndexer::FGraphIndexer()
{

}

FGraphIndexer::FGraphIndexer(const UEdGraph* GraphToIndex)
    : LinkedGraph(const_cast<UEdGraph*>(GraphToIndex))
{
    BuildGraphIndex();

    OnGraphChangedHandle = LinkedGraph->AddOnGraphChangedHandler(
        FOnGraphChanged::FDelegate::CreateRaw(this, &FGraphIndexer::OnGraphEvent)
    );
}

FGraphIndexer::~FGraphIndexer()
{
    IndexMap.Empty();
    NodeSet.Empty();
    AvailableIndices.Empty();

    LinkedGraph->RemoveOnGraphChangedHandler(OnGraphChangedHandle);

    LinkedGraph = nullptr;
}

bool FGraphIndexer::ContainsKey(const int& InKey)
{
    return IndexMap.Contains(InKey);
}

int FGraphIndexer::ContainsNode(UEdGraphNode* InNode)
{
    check(InNode != nullptr);

    if (!InNode->IsValidLowLevelFast() || !NodeSet.Contains(InNode->GetUniqueID()))
        return -1;

    const int* ReturnedIndex = IndexMap.FindKey(InNode);

    if (ReturnedIndex != nullptr)
    {
        return *ReturnedIndex;
    }
    else return -1;
}

void FGraphIndexer::ContainsNode(UEdGraphNode* InNode, int& OutIndex)
{
    OutIndex = ContainsNode(InNode);
}

int FGraphIndexer::GetKey(const UEdGraphNode* InNode)
{
    check(InNode != nullptr);

    if (!InNode->IsValidLowLevelFast())
        return -1;

    const int* FoundKey = IndexMap.FindKey(const_cast<UEdGraphNode*>(InNode));

    if (FoundKey != nullptr) return *FoundKey;
    else return -1;
}

bool FGraphIndexer::GetKey(const UEdGraphNode* InNode, int& OutKey)
{
    check(InNode != nullptr);

    if (!InNode->IsValidLowLevelFast())
        return false;

    const int* FoundKey = IndexMap.FindKey(const_cast<UEdGraphNode*>(InNode));
    if (FoundKey != nullptr)
    {
        OutKey = *FoundKey;
        return true;
    }
    else return false;
}

UEdGraphNode* FGraphIndexer::GetNode(const int& InIndex)
{
    if (!IndexMap.Contains(InIndex))
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided Index is not recognised"))

        return nullptr;
    }

    return IndexMap[InIndex];
}

void FGraphIndexer::GetPin(const int& InNodeIndex, const int& InPinIndex, UEdGraphPin* OutPin)
{
    UEdGraphNode* Node = GetNode(InNodeIndex);
    if (Node == nullptr)
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Requested Node at index %d is not valid."),
 InNodeIndex);
        return;
    }

    OutPin = Node->GetPinAt(InPinIndex); // Returns nullptr if invalid
}

UEdGraphPin* FGraphIndexer::GetPin(const int& InNodeIndex, const int& InPinIndex)
{
    UEdGraphNode* Node = GetNode(InNodeIndex);
    if (Node == nullptr)
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Requested Node at index %d is not valid."),
 InNodeIndex);
        return nullptr;
    }

    return Node->GetPinAt(InPinIndex); // Returns nullptr if invalid
}

void FGraphIndexer::GetNode(const int& InIndex, UEdGraphNode* OutNode)
{
    OutNode = GetNode(InIndex);
}

int FGraphIndexer::AddNode(const UEdGraphNode* InNode)
{
    check (InNode != nullptr);

    if (!InNode->IsValidLowLevelFast())
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided Node is not valid."))
    }

    int Index = ContainsNode(const_cast<UEdGraphNode*>(InNode));
    if (Index != -1)
    {
        return Index;
    }

    GetAvailableIndex(Index);

    NodeSet.Add(InNode->GetUniqueID());
    IndexMap.Add(Index, const_cast<UEdGraphNode*>(InNode));

    return Index;
}

void FGraphIndexer::AddNode(int& OutIndex, const UEdGraphNode& InNode)
{
    OutIndex = AddNode(&InNode);
}

int FGraphIndexer::GetOrAddNode(const UEdGraphNode* InNode)
{
    int Key = GetKey(InNode);
    if (Key != -1)
    {
        return Key;
    }

    return AddNode(InNode);
}

void FGraphIndexer::GetOrAddNode(const UEdGraphNode* InNode, int& OutIndex)
{
    OutIndex = GetKey(InNode);
    if (OutIndex != -1)
    {
        return;
    }

    OutIndex = AddNode(InNode);
}

void FGraphIndexer::RemoveNode(const int& InIndex)
{
    if (!IndexMap.Contains(InIndex))
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided Index is not recognised"))
    }

    const UEdGraphNode* Node = IndexMap[InIndex];

    if (Node->IsValidLowLevelFast())
    {
        NodeSet.Remove(Node->GetUniqueID());
        IndexMap.Remove(InIndex);
        AvailableIndices.Enqueue(InIndex);
    }
    else
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Stored Node in IndexMap is not vaild."))
    }
}

void FGraphIndexer::RemoveNode(const UEdGraphNode* InNode)
{
    check(InNode != nullptr);

    int Key = GetKey(InNode);
    if (Key == -1)
    {
        UE_LOG(LogOpenAccessibility, Warning, TEXT("Node does not exist in IndexMap."))
        return;
    }

    RemoveNode(Key);
}



// ------------------------
//  Graph Events
// ------------------------

void FGraphIndexer::OnGraphEvent(const FEdGraphEditAction& InAction)
{
    if (InAction.Graph != LinkedGraph)
    {
        return;
    }

    switch (InAction.Action)
    {
        case EEdGraphActionType::GRAPHACTION_AddNode:
        {
            for (const UEdGraphNode* Node : InAction.Nodes)
            {
                AddNode(Node);
            }

            break;
        }

        case EEdGraphActionType::GRAPHACTION_RemoveNode:
        {
            for (const UEdGraphNode* Node : InAction.Nodes)
            {
                RemoveNode(Node);
            }

            break;
        }
    }
}

void FGraphIndexer::OnGraphRebuild()
{
    IndexMap.Reset();
    NodeSet.Reset();
    AvailableIndices.Empty();

    BuildGraphIndex();
}

int FGraphIndexer::GetAvailableIndex()
{
    if (!AvailableIndices.IsEmpty())
    {
        int Index;
        if (AvailableIndices.Dequeue(Index))
            return Index;
    }

    return IndexMap.Num();
}

void FGraphIndexer::GetAvailableIndex(int& OutIndex)
{
    if (!AvailableIndices.IsEmpty() && AvailableIndices.Dequeue(OutIndex))
    {
        return;
    }
    else OutIndex = IndexMap.Num();
}

void FGraphIndexer::BuildGraphIndex()
{
    if (LinkedGraph == nullptr)
        return;

    for (TObjectPtr<UEdGraphNode> Node : LinkedGraph->Nodes)
    {
        AddNode(Node);
    }
}
```

5.21 OAccessibilityNodeFactory.cpp

```
// Fill out your copyright notice in the Description page of Project Settings.


#include "OAccessibilityNodeFactory.h"
#include "OpenAccessibilityLogging.h"

#include "Logging/StructuredLog.h"

#include "NodeFactory.h"
#include "EdGraphUtilities.h"

#include "Styling/AppStyle.h"
#include "SGraphPanel.h"
#include "SNodePanel.h"
#include "SGraphNode.h"
#include "SGraphPin.h"
#include "Widgets/SBoxPanel.h"
```

```
#include "Widgets/Text/STextBlock.h"

#include "OpenAccessibility.h"
#include "AccessibilityWidgets/SIndexer.h"

FAccessibilityNodeFactory::FAccessibilityNodeFactory() : FGraphPanelNodeFactory()
{
    UE_LOGFMT(LogOpenAccessibility, Display, "Accessibility Node Factory Constructed");
}

FAccessibilityNodeFactory::~FAccessibilityNodeFactory()
{

}

TSharedPtr<class SGraphNode> FAccessibilityNodeFactory::CreateNode(UEdGraphNode* InNode) const
{
    UE_LOG(LogOpenAccessibility, Display, TEXT("Accessibility Node Factory Used to construct %s"),
 *InNode->GetName());

    check(InNode);

    // Hack to get around the possible infinite loop of using
    // this factory to create the node from the factory itself.
    FEdGraphUtilities::UnregisterVisualNodeFactory(FOpenAccessibilityModule::Get().AccessibilityNodeFactory);
    TSharedPtr<SGraphNode> OutNode = FNodeFactory::CreateNodeWidget(InNode);
    FEdGraphUtilities::RegisterVisualNodeFactory(FOpenAccessibilityModule::Get().AccessibilityNodeFactory);

    // Get Node Accessibility Index, from registry
    TSharedRef<FGraphIndexer> GraphIndexer = FOpenAccessibilityModule::Get()
        .AssetAccessibilityRegistry->GetGraphIndexer(InNode->GetGraph());

    int NodeIndex = -1;
    GraphIndexer->GetOrAddNode(InNode, NodeIndex);

    {
        // Create Accessibility Widgets For Pins
        TArray<UEdGraphPin*> Pins = InNode->GetAllPins();
        TSharedPtr<SGraphPin> PinWidget;

        for (int i = 0; i < Pins.Num(); i++)
        {
            UEdGraphPin* Pin = Pins[i];

            PinWidget = OutNode->FindWidgetForPin(Pin);
            if (!PinWidget.IsValid())
            {
                continue;
            }

            WrapPinWidget(Pin, PinWidget.ToSharedRef(), i, OutNode.Get());
        }

        PinWidget.Reset();
    }

    // Wrap The Node Widget
    WrapNodeWidget(InNode, OutNode.ToSharedRef(), NodeIndex);

    return OutNode;
}

void FAccessibilityNodeFactory::WrapNodeWidget(UEdGraphNode* Node, TSharedRef<SGraphNode> NodeWidget,
 int NodeIndex) const
{
    TSharedRef<SWidget> WidgetToWrap = NodeWidget->GetSlot(ENodeZone::Center)->GetWidget();
    check(WidgetToWrap != SNullWidget::NullWidget);

    NodeWidget->GetOrAddSlot(ENodeZone::Center)
        .HAlign(HAlign_Fill)
        [
            SNew(SVerticalBox)

                + SVerticalBox::Slot()
                .HAlign(HAlign_Fill)
                .AutoHeight()
                .Padding(FMargin(1.5f, 0.25f))
                [
                    SNew(SOverlay)

                        + SOverlay::Slot()
                        [
                            SNew(SImage)
                                .Image(FAppStyle::Get().GetBrush("Graph.Node.Body"))
                        ]

                        + SOverlay::Slot()
                        .Padding(FMargin(4.0f, 0.0f))
                        [
                            SNew(SHorizontalBox)
                                + SHorizontalBox::Slot()
                                .HAlign(HAlign_Right)
                                .VAlign(VAlign_Center)
                                .Padding(1.f)
                                [
                                    SNew(SOverlay)
                                        + SOverlay::Slot()
                                        [
                                            SNew(SIndexer)
                                            .IndexValue(NodeIndex)
                                            .TextColor(FLinearColor::White)
                                            .BorderColor(FLinearColor::Gray)
                                        ]
                                ]
                        ]
                ]

                + SVerticalBox::Slot()
                .HAlign(HAlign_Fill)
                .AutoHeight()
                [
                    WidgetToWrap
                ]
        ];
}

void FAccessibilityNodeFactory::WrapPinWidget(UEdGraphPin* Pin, TSharedRef<SGraphPin> PinWidget, int PinIndex, SGraphNode* OwnerNode) const
{
    TSharedRef<SWidget> PinWidgetContent = PinWidget->GetContent();
    check(PinWidgetContent != SNullWidget::NullWidget);

    TSharedRef<SWidget> AccessibilityWidget = SNew(SOverlay)
        .Visibility_Lambda([OwnerNode]() -> EVisibility {

            if (OwnerNode->HasAnyUserFocusOrFocusedDescendants() || OwnerNode->IsHovered() || OwnerNode->GetOwnerPanel()->SelectionManager.IsNodeSelected(OwnerNode->GetNodeObj()))
                return EVisibility::Visible;

            return EVisibility::Hidden;
        })
        + SOverlay::Slot()
        [
            SNew(SIndexer)
            .IndexValue(PinIndex)
            .TextColor(FLinearColor::White)
            .BorderColor(FLinearColor::Gray)
        ];

    switch (Pin->Direction)
    {
        case EEdGraphPinDirection::EGPD_Input:
        {
            PinWidget->SetContent(
                SNew(SHorizontalBox)
                    + SHorizontalBox::Slot()
                    .AutoWidth()
                    [
                        PinWidgetContent
                    ]
                    + SHorizontalBox::Slot()
                    .AutoWidth()
                    [
                        AccessibilityWidget
                    ]
            );

            break;
        }

        case EEdGraphPinDirection::EGPD_Output:
        {
            PinWidget->SetContent(
                SNew(SHorizontalBox)
                    + SHorizontalBox::Slot()
                    .AutoWidth()
                    [
                        AccessibilityWidget
                    ]
                    + SHorizontalBox::Slot()
                    .AutoWidth()
                    [
                        PinWidgetContent
                    ]
            );
            break;
        }

        default:
        {
            UE_LOG(LogOpenAccessibility, Error, TEXT("Pin Direction Not Recognized"));
            break;
        }
    }
}
```

5.22 OAEditorAccessibilityManager.cpp

```
// Copyright F-Dudley. All Rights Reserved.


#include "OAEditorAccessibilityManager.h"

OAEditorAccessibilityManager::OAEditorAccessibilityManager()
{
}

OAEditorAccessibilityManager::~OAEditorAccessibilityManager()
{
}
```

5.23 OpenAccessibility.cpp

```
// Copyright Epic Games, Inc. All Rights Reserved.

#include "OpenAccessibility.h"
#include "OpenAccessibilityCommunication.h"
#include "OpenAccessibilityLogging.h"

#include "PhraseTree/PhraseNode.h"
#include "PhraseTree/PhraseInputNode.h"
#include "PhraseTree/PhraseStringInputNode.h"
#include "PhraseTree/PhraseDirectionalInputNode.h"
#include "PhraseTree/PhraseContextNode.h"
#include "PhraseTree/PhraseContextMenuNode.h"
#include "PhraseTree/PhraseEventNode.h"

#include "PhraseEvents/LocalizedInputLibrary.h"
#include "PhraseEvents/WindowInteractionLibrary.h"
#include "PhraseEvents/ViewInteractionLibrary.h"
#include "PhraseEvents/NodeInteractionLibrary.h"

#include "TranscriptionVisualizer.h"
#include "AccessibilityWrappers/AccessibilityAddNodeContextMenu.h"
#include "AccessibilityWrappers/AccessibilityGraphLocomotionContext.h"

#include "GraphActionNode.h"
#include "SGraphPanel.h"
#include "AccessibilityWrappers/AccessibilityGraphEditorContext.h"
#include "Widgets/Text/SMultiLineEditableText.h"
#include "Widgets/Input/SSearchBox.h"

#include "Framework/Docking/TabManager.h"
#include "Logging/StructuredLog.h"

#define LOCTEXT_NAMESPACE "FOpenAccessibilityModule"

void FOpenAccessibilityModule::StartupModule()
{
    UE_LOG(LogOpenAccessibility, Display, TEXT("OpenAccessibilityModule::StartupModule()"));

    // Create the Asset Registry
    AssetAccessibilityRegistry = MakeShared<FAssetAccessibilityRegistry, ESPMode::ThreadSafe>();

    // Register the Accessibility Node Factory
    AccessibilityNodeFactory = MakeShared<FAccessibilityNodeFactory, ESPMode::ThreadSafe>();
    FEdGraphUtilities::RegisterVisualNodeFactory(AccessibilityNodeFactory);

    // Construct Base Phrase Tree Libraries
    FOpenAccessibilityCommunicationModule::Get()
    .PhraseTreeUtils->RegisterFunctionLibrary(
        NewObject<ULocalizedInputLibrary>()
    );

    FOpenAccessibilityCommunicationModule::Get()
    .PhraseTreeUtils->RegisterFunctionLibrary(
        NewObject<UWindowInteractionLibrary>()
    );

    FOpenAccessibilityCommunicationModule::Get()
    .PhraseTreeUtils->RegisterFunctionLibrary(
        NewObject<UViewInteractionLibrary>()
    );

    FOpenAccessibilityCommunicationModule::Get()
    .PhraseTreeUtils->RegisterFunctionLibrary(
        NewObject<UNodeInteractionLibrary>()
    );

    CreateTranscriptionVisualization();

    // Register Console Commands
    RegisterConsoleCommands();
}

void FOpenAccessibilityModule::ShutdownModule()
{
    UE_LOG(LogOpenAccessibility, Display, TEXT("OpenAccessibilityModule::ShutdownModule()"));

    UnregisterConsoleCommands();
}

void FOpenAccessibilityModule::CreateTranscriptionVisualization()
{
    TranscriptionVisualizer = MakeShared<FTranscriptionVisualizer, ESPMode::ThreadSafe>();

    FOpenAccessibilityCommunicationModule::Get().OnTranscriptionRecieved
        .AddSP(TranscriptionVisualizer.ToSharedRef(),
&FTranscriptionVisualizer::OnTranscriptionRecieved);
}

void FOpenAccessibilityModule::RegisterConsoleCommands()
{
    ConsoleCommands.Add(IConsoleManager::Get().RegisterConsoleCommand(
        TEXT("OpenAccessibility.Debug.SendPhraseEvent"),
        TEXT("Sends the provided Phrase to the Phrase Tree, replicating the STT Communication Module's
Transcription Recieving."),
        FConsoleCommandWithArgsDelegate::CreateLambda([this](const TArray<FString> &Args) {
            if (Args.Num() == 0)
                return;

            FString ProvidedPhrase;
            for (const FString& Arg : Args)
            {
                ProvidedPhrase += Arg + TEXT(" ");
            }

            ProvidedPhrase.TrimStartAndEndInline();
            ProvidedPhrase.ToUpperInline();

            FOpenAccessibilityCommunicationModule::Get()
                .OnTranscriptionRecieved.Broadcast(TArray<FString>{ ProvidedPhrase });
        }),

        ECVF_Default
    ));

    ConsoleCommands.Add(IConsoleManager::Get().RegisterConsoleCommand(
        TEXT("OpenAccessibility.Debug.LogActiveIndexes"),
        TEXT("Logs the Active Indexes of the Active Tab"),

        FConsoleCommandDelegate::CreateLambda([this]() {

            TSharedPtr<SDockTab> ActiveTab = FGlobalTabmanager::Get()->GetActiveTab();
            SGraphEditor* ActiveGraphEditor =
(SGraphEditor*)ActiveTab->GetContent().ToSharedPtr().Get();
            if (ActiveGraphEditor == nullptr)
            {
                UE_LOG(LogOpenAccessibility, Display, TEXT("Active Tab Not SGraphEditor"));
                return;
            }


            TSharedRef<FGraphIndexer> GraphIndexer =
AssetAccessibilityRegistry->GetGraphIndexer(ActiveGraphEditor->GetCurrentGraph());
        }),

        ECVF_Default
```

```
   ));

   ConsoleCommands.Add(IConsoleManager::Get().RegisterConsoleCommand(
       TEXT("OpenAccessibility.Debug.OpenAccessibilityGraph_AddNodeMenu"),
       TEXT("Opens the context menu for adding nodes for the active graph editor."),

       FConsoleCommandDelegate::CreateLambda(
           [this]() {

               TSharedPtr<SGraphEditor> ActiveGraphEditor = nullptr;
               {
                   // Getting Graph Editor Section

                   TSharedPtr<SDockTab> ActiveTab = FGlobalTabmanager::Get()->GetActiveTab();
                   if (!ActiveTab.IsValid())
                       return;

                   ActiveGraphEditor = StaticCastSharedPtr<SGraphEditor>(ActiveTab->GetContent().ToSharedPtr());
                   if (!ActiveGraphEditor.IsValid())
                   {
                       UE_LOG(LogOpenAccessibility, Display, TEXT("Active Tab Not SGraphEditor"));
                       return;
                   }
               }

               TSharedPtr<IMenu> Menu;
               TSharedPtr<SWindow> MenuWindow;
               TSharedPtr<SGraphActionMenu> GraphActionMenu;
               TSharedPtr<SSearchBox> SearchBox;
               TSharedPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeView;
               {
                   // Summoning Create Node Menu Section
                   // and Getting any Key Widgets

                   ActiveGraphEditor->GetGraphPanel()->SummonCreateNodeMenuFromUICommand(0);

                   TSharedPtr<SWidget> KeyboardFocusedWidget = StaticCastSharedPtr<SEditableText>(FSlateApplication::Get().GetKeyboardFocusedWidget());
                   if (!KeyboardFocusedWidget.IsValid())
                   {
                       UE_LOG(LogOpenAccessibility, Display, TEXT("Cannot get Keyboard Focused Widget."));
                       return;
                   }

                   UE_LOG(LogOpenAccessibility, Display, TEXT("Keyboard Focused Widget Type: %s"), *KeyboardFocusedWidget->GetTypeAsString());

                   // Getting Menu Object
                   FWidgetPath KeyboardFocusedWidgetPath;
                   if (FSlateApplication::Get().FindPathToWidget(KeyboardFocusedWidget.ToSharedRef(), KeyboardFocusedWidgetPath))
                   {
                       UE_LOG(LogOpenAccessibility, Display, TEXT("Keyboard Focused Widget Path Found."));
                   }
                   else return;

                   Menu = FSlateApplication::Get().FindMenuInWidgetPath(KeyboardFocusedWidgetPath);

                   // Getting Graph Action Menu Object
                   GraphActionMenu = StaticCastSharedPtr<SGraphActionMenu>(
                       KeyboardFocusedWidget
                           ->GetParentWidget()
                           ->GetParentWidget()
                           ->GetParentWidget()
                           ->GetParentWidget()
                           ->GetParentWidget()
                   );

                   SearchBox = StaticCastSharedPtr<SSearchBox>(
                       KeyboardFocusedWidget
                           ->GetParentWidget()
                           ->GetParentWidget()
                           ->GetParentWidget()
                   );

                   TSharedRef<SWidget> SearchBoxSibling = SearchBox->GetParentWidget()->GetChildren()->GetChildAt(1);
                   TreeView = StaticCastSharedRef<STreeView<TSharedPtr<FGraphActionNode»>(
                       SearchBoxSibling->GetChildren()->GetChildAt(0)->GetChildren()->GetChildAt(0)
                   );

                   UE_LOG(LogOpenAccessibility, Log, TEXT("THIS IS THE STRING: %s"), *TreeView->GetTypeAsString());

```

```cpp
                MenuWindow = FSlateApplication::Get().FindWidgetWindow(KeyboardFocusedWidget.ToSharedRef());
            }

            UAccessibilityAddNodeContextMenu* AddNodeContextMenu = NewObject<UAccessibilityAddNodeContextMenu>();
            AddNodeContextMenu->AddToRoot();
            AddNodeContextMenu->Init(
                Menu.ToSharedRef(),
                FOpenAccessibilityCommunicationModule::Get().PhraseTree->AsShared()
            );

            AddNodeContextMenu->ScaleMenu(1.5f);

            FSlateApplication::Get().SetKeyboardFocus(TreeView);

            FPhraseTreeContextManager& ContextManager =FOpenAccessibilityCommunicationModule::Get()
                  .PhraseTree->GetContextManager();

            ContextManager.PushContextObject(AddNodeContextMenu);
        }),

    ECVF_Default
));

ConsoleCommands.Add(IConsoleManager::Get().RegisterConsoleCommand(
    TEXT("OpenAccessibility.Debug.OpenAccessibilityGraph_GenericContextMenu"),
    TEXT("Opens the Context Menu for the Active Graph Editor, and Uses Generic Bindings For Commands"),

    FConsoleCommandDelegate::CreateLambda(
        [this]()
        {
            TSharedPtr<SGraphEditor> ActiveGraphEditor = nullptr;
            {
                // Getting Graph Editor Section

                TSharedPtr<SDockTab> ActiveTab = FGlobalTabmanager::Get()->GetActiveTab();
                if (!ActiveTab.IsValid())
                    return;

                ActiveGraphEditor = StaticCastSharedPtr<SGraphEditor>(ActiveTab->GetContent().ToSharedPtr());
                if (!ActiveGraphEditor.IsValid() && ActiveGraphEditor->GetType() == "SGraphEditor")
                {
                    UE_LOG(LogOpenAccessibility, Display, TEXT("Active Tab Not SGraphEditor"));
                    return;
                }
            }

            SGraphPanel* ActiveGraphPanel = ActiveGraphEditor->GetGraphPanel();

            FVector2D SpawnLocation;
            {
                TSharedPtr<SWindow> TopLevelWindow = FSlateApplication::Get().GetActiveTopLevelRegularWindow();
                if (TopLevelWindow.IsValid())
                {
                    SpawnLocation = TopLevelWindow->GetPositionInScreen();
                    FVector2D WindowSize = TopLevelWindow->GetSizeInScreen();

                    SpawnLocation.X += WindowSize.X / 5;
                    SpawnLocation.Y += WindowSize.Y / 5;
                }
                else
                {
                    FDisplayMetrics DisplayMetrics;
                    FSlateApplication::Get().GetDisplayMetrics(DisplayMetrics);

                    SpawnLocation = FVector2D(
                        DisplayMetrics.PrimaryDisplayWidth / 5,
                        DisplayMetrics.PrimaryDisplayHeight / 5
                    );
                }
            }

            TSharedPtr<SWidget> ContextWidgetToFocus = ActiveGraphPanel->SummonContextMenu(
                SpawnLocation,
                ActiveGraphPanel->GetPastePosition(),
                nullptr,
                nullptr,
                TArray<UEdGraphPin*>()
            );

            FWidgetPath ContextWidgetToFocusPath;
```

```
                if (FSlateApplication::Get().FindPathToWidget(ContextWidgetToFocus.ToSharedRef(), ContextWidgetToFocusPath))
                {
                    UAccessibilityGraphEditorContext* GraphContext = NewObject<UAccessibilityGraphEditorContext>();
                    GraphContext->AddToRoot();

                    GraphContext->Init(
                        FSlateApplication::Get().FindMenuInWidgetPath(ContextWidgetToFocusPath).ToSharedRef(),
                        FOpenAccessibilityCommunicationModule::Get().PhraseTree->AsShared()

                    );

                    GraphContext->ScaleMenu(1.5f);
                }

            }
        )
    )),

    ConsoleCommands.Add(IConsoleManager::Get().RegisterConsoleCommand(
        TEXT("OpenAccessibility.Debug.OpenAccessibilityGraph_SummonImprovedLocomotion"),
        TEXT("Summons the Improved Locomotion Menu for the Active Graph Editor."),

        FConsoleCommandDelegate::CreateLambda(
            [this]() {
                TSharedPtr<SGraphEditor> ActiveGraphEditor = nullptr;
                {
                    // Getting Graph Editor Section

                    TSharedPtr<SDockTab> ActiveTab = FGlobalTabmanager::Get()->GetActiveTab();
                    if (!ActiveTab.IsValid())
                        return;

                    ActiveGraphEditor = StaticCastSharedPtr<SGraphEditor>(ActiveTab->GetContent().ToSharedPtr());
                    if (!ActiveGraphEditor.IsValid() || ActiveGraphEditor->GetTypeAsString() != "SGraphEditor")
                    {
                        UE_LOG(LogOpenAccessibility, Display, TEXT("Active Tab Not SGraphEditor"));
                        return;
                    }
                }

                UAccessibilityGraphLocomotionContext* LocomotionContext = NewObject<UAccessibilityGraphLocomotionContext>();
                LocomotionContext->AddToRoot();
                LocomotionContext->Init(ActiveGraphEditor.ToSharedRef());

                FPhraseTreeContextManager& ContextManager = FOpenAccessibilityCommunicationModule::Get()
                  .PhraseTree->GetContextManager();

                ContextManager.PushContextObject(LocomotionContext);
            }),

        ECVF_Default
    ));
}

void FOpenAccessibilityModule::UnregisterConsoleCommands()
{
    IConsoleCommand* ConsoleCommand = nullptr;
    while (ConsoleCommands.Num() > 0)
    {
        ConsoleCommand = ConsoleCommands.Pop();

        IConsoleManager::Get().UnregisterConsoleObject(ConsoleCommand);

        delete ConsoleCommand;
        ConsoleCommand = nullptr;
    }
}

#undef LOCTEXT_NAMESPACE

IMPLEMENT_MODULE(FOpenAccessibilityModule, OpenAccessibility)
```

5.24 LocalizedInputLibrary.cpp

```
#include "PhraseEvents/LocalizedInputLibrary.h"

```

```
#include "ToolContextInterfaces.h"
#include "PhraseEvents/Utils.h"

#include "PhraseTree/PhraseStringInputNode.h"
#include "PhraseTree/PhraseEventNode.h"

#include "PhraseTree/Containers/Input/UParseStringInput.h"
#include "PhraseTree/Containers/Input/UParseIntInput.h"
#include "Widgets/Text/SMultiLineEditableText.h"

ULocalizedInputLibrary::ULocalizedInputLibrary(const FObjectInitializer &ObjectInitializer)
{

}

ULocalizedInputLibrary::~ULocalizedInputLibrary()
{

}

void ULocalizedInputLibrary::BindBranches(TSharedRef<FPhraseTree> PhraseTree)
{
    PhraseTree->BindBranch(
        MakeShared<FPhraseNode>(TEXT("INPUT"),
        TPhraseNodeArray {

            MakeShared<FPhraseNode>(TEXT("ADD"),
            TPhraseNodeArray {

                MakeShared<FPhraseStringInputNode>(TEXT("PHRASE_TO_ADD"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
 &ULocalizedInputLibrary::KeyboardInputAdd))

                })

            }),

            MakeShared<FPhraseNode>(TEXT("REMOVE"),
            TPhraseNodeArray {

                MakeShared<FPhraseInputNode<int32»(TEXT("AMOUNT"),
                TPhraseNodeArray {


 MakeShared<FPhraseEventNode>(CreateParseDelegate(this,&ULocalizedInputLibrary::KeyboardInputRemove))

                })

            }),

            MakeShared<FPhraseNode>(TEXT("RESET"),
            TPhraseNodeArray {

                MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
 &ULocalizedInputLibrary::KeyboardInputReset))

            }),

            /*
            MakeShared<FPhraseNode>(TEXT("CONFIRM"),
            TPhraseNodeArray {

                MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
 &ULocalizedInputLibrary::KeyboardInputConfirm))

            }),
            */

            MakeShared<FPhraseNode>(TEXT("EXIT"),
            TPhraseNodeArray {

                MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
 &ULocalizedInputLibrary::KeyboardInputExit))

            })

        })
    );
}

void ULocalizedInputLibrary::KeyboardInputAdd(FParseRecord &Record) {
    GET_ACTIVE_KEYBOARD_WIDGET(KeyboardFocusedWidget);

    FString WidgetType = KeyboardFocusedWidget->GetTypeAsString();

    UParseStringInput *PhraseInput = Record.GetPhraseInput<UParseStringInput>(TEXT("PHRASE_TO_ADD"));
    if (PhraseInput == nullptr)
        return;

    if (WidgetType == "SEditableText")
    {
        TSharedPtr<SEditableText> EditableText = StaticCastSharedPtr<SEditableText>(KeyboardFocusedWidget);
        if (!EditableText.IsValid()) {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputAdd: CURRENT ACTIVE WIDGET IS NOT OF TYPE - SEditableText"));
            return;
        }

        FString CurrText = EditableText->GetText().ToString();
        EditableText->SetText(
            FText::FromString(CurrText.TrimStartAndEnd() + TEXT(" ") + PhraseInput->GetValue())
        );
    }
    else if (WidgetType == "SMultiLineEditableText")
    {
        TSharedPtr<SMultiLineEditableText> MultiLineEditableText = StaticCastSharedPtr<SMultiLineEditableText>(KeyboardFocusedWidget);
        if (!MultiLineEditableText.IsValid()) {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputAdd: CURRENT ACTIVE WIDGET IS NOT OF TYPE - SMultiLineEditableText"));
            return;
        }

        FString CurrText = MultiLineEditableText->GetText().ToString();
        MultiLineEditableText->SetText(
            FText::FromString(CurrText.TrimStartAndEnd() + TEXT(" ") + PhraseInput->GetValue())
        );
    }
    else UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputAdd: CURRENT ACTIVE WIDGET IS NOT AN INTERFACEABLE TYPE"));
}

void ULocalizedInputLibrary::KeyboardInputRemove(FParseRecord& Record)
{
    GET_ACTIVE_KEYBOARD_WIDGET(KeyboardFocusedWidget);

    FString WidgetType = KeyboardFocusedWidget->GetTypeAsString();

    UParseIntInput* RemoveInput = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (RemoveInput == nullptr)
        return;

    if (WidgetType == "SEditableText")
    {
        TSharedPtr<SEditableText> EditableText = StaticCastSharedPtr<SEditableText>(KeyboardFocusedWidget);
        if (!EditableText.IsValid()) {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputRemove: CURRENT ACTIVE WIDGET IS NOT OF TYPE - SEditableText"));
            return;
        }

        EditableText->SetText(
            FText::FromString(
                EventUtils::RemoveWordsFromEnd(EditableText->GetText().ToString(), RemoveInput->GetValue())
            )
        );
    }
    else if (WidgetType == "SMultiLineEditableText")
    {
        TSharedPtr<SMultiLineEditableText> MultiLineEditableText = StaticCastSharedPtr<SMultiLineEditableText>(KeyboardFocusedWidget);
        if (!MultiLineEditableText.IsValid()) {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputRemove: CURRENT ACTIVE WIDGET IS NOT OF TYPE - SMultiLineEditableText"));
            return;
        }

        MultiLineEditableText->SetText(
            FText::FromString(
                EventUtils::RemoveWordsFromEnd(MultiLineEditableText->GetText().ToString(), RemoveInput->GetValue())
            )
        );
    }
    else UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputRemove: CURRENT ACTIVE WIDGET IS NOT AN INTERFACEABLE TYPE"));
}

void ULocalizedInputLibrary::KeyboardInputReset(FParseRecord &Record)
```

```
{
    GET_ACTIVE_KEYBOARD_WIDGET(KeyboardFocusedWidget);

    FString WidgetType = KeyboardFocusedWidget->GetTypeAsString();

    if (WidgetType == "SEditableText")
    {
        TSharedPtr<SEditableText> EditableText =
 StaticCastSharedPtr<SEditableText>(KeyboardFocusedWidget);
        if (!EditableText.IsValid()) {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputReset: CURRENT ACTIVE
 WIDGET IS NOT OF TYPE - SEditableText"));
            return;
        }

        EditableText->SetText(
            FText::FromString(TEXT(""))
        );
    }
    else if (WidgetType == "SMultiLineEditableText")
    {
        TSharedPtr<SMultiLineEditableText> MultiLineEditableText =
 StaticCastSharedPtr<SMultiLineEditableText>(KeyboardFocusedWidget);
        if (!MultiLineEditableText.IsValid()) {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputReset: CURRENT ACTIVE
 WIDGET IS NOT OF TYPE - SMultiLineEditableText"));
            return;
        }

        MultiLineEditableText->SetText(
            FText::FromString(TEXT(""))
        );
    }
    else UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputReset: CURRENT ACTIVE
 WIDGET IS NOT AN INTERFACEABLE TYPE"));
}

void ULocalizedInputLibrary::KeyboardInputConfirm(FParseRecord& Record)
{
    GET_ACTIVE_KEYBOARD_WIDGET(KeyboardFocusedWidget);

    FName WidgetType = KeyboardFocusedWidget->GetType();

    if (WidgetType == SEditableText::StaticWidgetClass().GetWidgetType())
    {
        TSharedPtr<SEditableText> EditableText =
 StaticCastSharedPtr<SEditableText>(KeyboardFocusedWidget);
        if (!EditableText.IsValid())
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputConfirm: CURRENT
 ACTIVE WIDGET IS NOT OF TYPE - SEditableText"))
            return;
        }

    }
    else if (WidgetType == SMultiLineEditableText::StaticWidgetClass().GetWidgetType())
    {
        TSharedPtr<SMultiLineEditableText> MultiLineEditableText =
 StaticCastSharedPtr<SMultiLineEditableText>(KeyboardFocusedWidget);
        if (!MultiLineEditableText.IsValid())
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputConfirm: CURRENT
 ACTIVE WIDGET IS NOT OF TYPE - SMultiLineEditableText"))
            return;
        }

    }
    else UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("KeyboardInputConfirm: CURRENT ACTIVE
 WIDGET IS NOT AN INTERFACEABLE TYPE"))
}

void ULocalizedInputLibrary::KeyboardInputExit(FParseRecord &Record)
{
    FSlateApplication& SlateApp = FSlateApplication::Get();
    if (!SlateApp.IsInitialized())
        return;

    SlateApp.ClearKeyboardFocus();
}
```

5.25 NodeInteractionLibrary.cpp

```
#include "PhraseEvents/NodeInteractionLibrary.h"
```

```
#include "PhraseEvents/Utils.h"

#include "BlueprintEditor.h"
#include "SNodePanel.h"
#include "SGraphPanel.h"
#include "Kismet2/KismetEditorUtilities.h"
#include "Kismet2/BlueprintEditorUtils.h"

#include "PhraseTree/Containers/Input/InputContainers.h"
#include "AccessibilityWrappers/AccessibilityGraphEditorContext.h"
#include "AccessibilityWrappers/AccessibilityGraphLocomotionContext.h"

#include "PhraseTree/PhraseInputNode.h"
#include "PhraseTree/PhraseStringInputNode.h"
#include "PhraseTree/PhraseDirectionalInputNode.h"
#include "PhraseTree/PhraseContextNode.h"
#include "PhraseTree/PhraseContextMenuNode.h"
#include "PhraseTree/PhraseEventNode.h"

UNodeInteractionLibrary::UNodeInteractionLibrary(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{

}

UNodeInteractionLibrary::~UNodeInteractionLibrary()
{

}

void UNodeInteractionLibrary::BindBranches(TSharedRef<FPhraseTree> PhraseTree)
{
    // Events
    TDelegate<void(int32)> NodeIndexFocusDelegate = CreateInputDelegate(this,
 &UNodeInteractionLibrary::NodeIndexFocus);


    // Add Node Children Branch
    TPhraseNodeArray AddNodeContextChildren = TPhraseNodeArray {

        MakeShared<FPhraseNode>(TEXT("SELECT"),
        TPhraseNodeArray {

            MakeShared<FPhraseInputNode<int32»(TEXT("SELECTION_INDEX"),
            TPhraseNodeArray {

                MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
 &UNodeInteractionLibrary::NodeAddSelect))

            })

        }),

        MakeShared<FPhraseNode>(TEXT("SEARCH"),
        TPhraseNodeArray{

            MakeShared<FPhraseNode>(TEXT("ADD"),
            TPhraseNodeArray {

                MakeShared<FPhraseStringInputNode>(TEXT("SEARCH_PHRASE"),
                TPhraseNodeArray{

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
 &UNodeInteractionLibrary::NodeAddSearchAdd))

                })

            }),

            MakeShared<FPhraseNode>(TEXT("REMOVE"),
            TPhraseNodeArray {

                MakeShared<FPhraseInputNode<int32»(TEXT("AMOUNT"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
 &UNodeInteractionLibrary::NodeAddSearchRemove))

                })

            }),

            MakeShared<FPhraseNode>(TEXT("RESET"),
            TPhraseNodeArray {

                MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
 &UNodeInteractionLibrary::NodeAddSearchReset))
```

```
            })

        }),

        MakeShared<FPhraseNode>(TEXT("SCROLL"),
        TPhraseNodeArray {

            MakeShared<FPhraseScrollInputNode>(TEXT("DIRECTION"),
            TPhraseNodeArray {

                MakeShared<FPhraseInputNode<int32»(TEXT("AMOUNT"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::NodeAddScroll))

                })

            }),

        }),

    };

    PhraseTree->BindBranches(
        TPhraseNodeArray
        {
            MakeShared<FPhraseNode>(TEXT("NODE"),
            TPhraseNodeArray {

                MakeShared<FPhraseInputNode<int32»(TEXT("NODE_INDEX"),
                TPhraseNodeArray {

                    MakeShared<FPhraseNode>(TEXT("MOVE"),
                    TPhraseNodeArray {

                        MakeShared<FPhrase2DDirectionalInputNode>(TEXT("DIRECTION"),
                        TPhraseNodeArray {

                            MakeShared<FPhraseInputNode<int32»(TEXT("AMOUNT"),
                            TPhraseNodeArray {

                                MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::MoveNode))

                            })

                        })

                    }),

                    MakeShared<FPhraseNode>(TEXT("REMOVE"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::DeleteNode))

                    }),

                    MakeShared<FPhraseInputNode<int32>>(TEXT("PIN_INDEX"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseNode>(TEXT("CONNECT"),
                        TPhraseNodeArray {

                            MakeShared<FPhraseContextMenuNode<UAccessibilityGraphEditorContext»(
                                TEXT("ADD"),
                                1.5f,
                                CreateMenuDelegate(this, &UNodeInteractionLibrary::NodeAddPinMenu),
                                AddNodeContextChildren
                            ),

                            MakeShared<FPhraseInputNode<int32>>(TEXT("NODE_INDEX"),
                            TPhraseNodeArray {

                                MakeShared<FPhraseInputNode<int32»(TEXT("PIN_INDEX"),
                                TPhraseNodeArray {

                                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::PinConnect))

                                })

                            }, NodeIndexFocusDelegate)

                        }),

                        MakeShared<FPhraseNode>(TEXT("DISCONNECT"),
                        TPhraseNodeArray {

                            MakeShared<FPhraseInputNode<int32»(TEXT("NODE_INDEX"),
                            TPhraseNodeArray {

                                MakeShared<FPhraseInputNode<int32»(TEXT("PIN_INDEX"),
                                TPhraseNodeArray {

                                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::PinDisconnect))

                                })

                            })

                        })

                    })

                }, NodeIndexFocusDelegate),

                MakeShared<FPhraseNode>(TEXT("SELECT"),
                TPhraseNodeArray {

                    MakeShared<FPhraseInputNode<int32»(TEXT("NODE_INDEX"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::SelectionNodeToggle))

                    }),

                    MakeShared<FPhraseNode>(TEXT("RESET"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::SelectionReset))

                    }),

                    MakeShared<FPhraseNode>(TEXT("MOVE"),
                    TPhraseNodeArray {

                        MakeShared<FPhrase2DDirectionalInputNode>(TEXT("DIRECTION"),
                        TPhraseNodeArray {

                            MakeShared<FPhraseInputNode<int32»(TEXT("AMOUNT"),
                            TPhraseNodeArray {

                                MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::SelectionMove))

                            })

                        })

                    }),

                    MakeShared<FPhraseNode>(TEXT("STRAIGHTEN"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::SelectionStraighten))

                    }),

                    MakeShared<FPhraseNode>(TEXT("ALIGNMENT"),
                    TPhraseNodeArray {

                        MakeShared<FPhrasePositionalInputNode>(TEXT("POSITION"),
                        TPhraseNodeArray {

                            MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::SelectionAlignment))

                        })

                    }),

                    MakeShared<FPhraseNode>(TEXT("COMMENT"),
                    TPhraseNodeArray{

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this, &UNodeInteractionLibrary::SelectionComment))

                    })

                }),

                MakeShared<FPhraseContextMenuNode<UAccessibilityGraphEditorContext>>(
                    TEXT("ADD"),
                    1.5f,
                    CreateMenuDelegate(this, &UNodeInteractionLibrary::NodeAddMenu),
                    AddNodeContextChildren
                ),

            }),

            MakeShared<FPhraseNode>(TEXT("GRAPH"),
            TPhraseNodeArray {

                MakeShared<FPhraseNode>(TEXT("COMPILE"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::BlueprintCompile))

                }),

                MakeShared<FPhraseContextNode<UAccessibilityGraphLocomotionContext>>(TEXT("MOVE"),
                TPhraseNodeArray {

                    MakeShared<FPhraseNode>(TEXT("SELECT"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseInputNode<int32»(TEXT("INDEX"),
                        TPhraseNodeArray {

                            MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::LocomotionSelect))

                        })

                    }),

                    MakeShared<FPhraseNode>(TEXT("REVERT"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::LocomotionRevert))

                    }),

                    MakeShared<FPhraseNode>(TEXT("CONFIRM"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::LocomotionConfirm))

                    }),

                    MakeShared<FPhraseNode>(TEXT("CANCEL"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UNodeInteractionLibrary::LocomotionCancel))

                    })

                }),
            })
        }
    );

};


void UNodeInteractionLibrary::MoveNode(FParseRecord &Record) {
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UParseIntInput* IndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("NODE_INDEX"));
    UParseEnumInput* DirectionInput = Record.GetPhraseInput<UParseEnumInput>(TEXT("DIRECTION"));
    UParseIntInput* AmountInput = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (IndexInput == nullptr || DirectionInput == nullptr || AmountInput == nullptr)
        return;

    TSharedRef<FAssetAccessibilityRegistry> AssetRegistry = GetAssetRegistry();
    TSharedRef<FGraphIndexer> Indexer =
AssetRegistry->GetGraphIndexer(ActiveGraphEditor->GetCurrentGraph());

    UEdGraphNode* Node = Indexer->GetNode(IndexInput->GetValue());
    if (Node == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("MoveNode: Node Not Found"));
        return;
    }

    FVector2D PositionDelta = FVector2D::ZeroVector;
    switch (EPhrase2DDirectionalInput(DirectionInput->GetValue()))
    {
        case EPhrase2DDirectionalInput::UP:
            PositionDelta.Y -= AmountInput->GetValue();
            break;

        case EPhrase2DDirectionalInput::DOWN:
            PositionDelta.Y += AmountInput->GetValue();
            break;

        case EPhrase2DDirectionalInput::LEFT:
            PositionDelta.X -= AmountInput->GetValue();
            break;

        case EPhrase2DDirectionalInput::RIGHT:
            PositionDelta.X += AmountInput->GetValue();
            break;

        default:
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("MoveNode: Invalid Direction"));
            return;
    }

    SGraphPanel* GraphPanel = ActiveGraphEditor->GetGraphPanel();
    if (GraphPanel == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("MoveNode: Linked Graph Panel Not
 Found"));
    }

    TSharedPtr<SGraphNode> NodeWidget = GraphPanel ? GraphPanel->GetNodeWidgetFromGuid(Node->NodeGuid)
 : TSharedPtr<SGraphNode>();
    if (NodeWidget.IsValid())
    {
        SNodePanel::SNode::FNodeSet NodeFilter;
        NodeWidget->MoveTo(FVector2D(Node->NodePosX, Node->NodePosY) + PositionDelta, NodeFilter);
    }
    else
    {
        Node->Modify();
        Node->NodePosX += PositionDelta.X;
        Node->NodePosY += PositionDelta.Y;
    }

    // Move Comment Node Children
    // Note: This is a workaround for the MoveTo Function not calling the override in
 UEdGraphNode_Comment
    if (UEdGraphNode_Comment* CommentNode = Cast<UEdGraphNode_Comment>(Node))
    {
        for (UObject* _CommentChildNode : CommentNode->GetNodesUnderComment())
        {
            if (UEdGraphNode* CommentChildNode = Cast<UEdGraphNode>(_CommentChildNode))
            {
                if (!GraphPanel->SelectionManager.IsNodeSelected(CommentChildNode))
                {
                    CommentChildNode->Modify();
                    CommentChildNode->NodePosX += PositionDelta.X;
                    CommentChildNode->NodePosY += PositionDelta.Y;
                }
            }
        }
    }
}

void UNodeInteractionLibrary::DeleteNode(FParseRecord& Record)
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UParseIntInput* IndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("NODE_INDEX"));
    if (IndexInput == nullptr)
        return;

    TSharedRef<FAssetAccessibilityRegistry> AssetRegistry = GetAssetRegistry();
    TSharedRef<FGraphIndexer> Indexer =
 AssetRegistry->GetGraphIndexer(ActiveGraphEditor->GetCurrentGraph());

    UEdGraphNode* Node = Indexer->GetNode(IndexInput->GetValue());
    if (Node == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("DeleteNode: Node Not Found"));
        return;
    }

    Node->Modify();
    Node->DestroyNode();
}

void UNodeInteractionLibrary::NodeIndexFocus(int32 Index)
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    TSharedRef<FGraphIndexer> Indexer = GetAssetRegistry()->GetGraphIndexer(
        ActiveGraphEditor->GetCurrentGraph()
    );

    UEdGraphNode* Node = Indexer->GetNode(Index);
    if (Node == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeSelectionFocus: Node Not Found"));
        return;
    }

    ActiveGraphEditor->SetNodeSelection(Node, true);
}

void UNodeInteractionLibrary::PinConnect(FParseRecord& Record)
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UEdGraph* Graph = ActiveGraphEditor->GetCurrentGraph();

    TArray<UParseInput*> NodeInputs = Record.GetPhraseInputs(TEXT("NODE_INDEX"));
    TArray<UParseInput*> PinInputs = Record.GetPhraseInputs(TEXT("PIN_INDEX"));

    if (NodeInputs.Num() != 2 || PinInputs.Num() != 2)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("PinConnect: Invalid Inputs Amount"));
        return;
    }

    TSharedRef<FGraphIndexer> Indexer = GetAssetRegistry()->GetGraphIndexer(Graph);

    UEdGraphPin* SourcePin = Indexer->GetPin(
        Cast<UParseIntInput>(NodeInputs[0])->GetValue(),
        Cast<UParseIntInput>(PinInputs[0])->GetValue()
    );

    UEdGraphPin* TargetPin = Indexer->GetPin(
        Cast<UParseIntInput>(NodeInputs[1])->GetValue(),
        Cast<UParseIntInput>(PinInputs[1])->GetValue()
    );

    if (SourcePin == nullptr || TargetPin == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("PinConnect: Pins Not Found"));
        return;
    }

    if (!Graph->GetSchema()->TryCreateConnection(SourcePin, TargetPin))
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("PinConnect: Pin Connection Failed"));
    }
}

void UNodeInteractionLibrary::PinDisconnect(FParseRecord& Record)
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UEdGraph* Graph = ActiveGraphEditor->GetCurrentGraph();

    TArray<UParseInput*> NodeInputs = Record.GetPhraseInputs(TEXT("NODE_INDEX"));
    TArray<UParseInput*> PinInputs = Record.GetPhraseInputs(TEXT("PIN_INDEX"));

    if (NodeInputs.Num() != 2 || PinInputs.Num() != 2)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("PinDisconnect: Invalid Inputs
Amount"));
        return;
    }

    TSharedRef<FGraphIndexer> Indexer = GetAssetRegistry()->GetGraphIndexer(Graph);

    UEdGraphPin* SourcePin = Indexer->GetPin(
        Cast<UParseIntInput>(NodeInputs[0])->GetValue(),
        Cast<UParseIntInput>(PinInputs[0])->GetValue()
    );

    UEdGraphPin* TargetPin = Indexer->GetPin(
        Cast<UParseIntInput>(NodeInputs[1])->GetValue(),
        Cast<UParseIntInput>(PinInputs[1])->GetValue()
    );

    if (SourcePin == nullptr || TargetPin == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("PinDisconnect: Pins Not Found"));
        return;
    }

    Graph->GetSchema()->BreakSinglePinLink(SourcePin, TargetPin);
}

TSharedPtr<IMenu> UNodeInteractionLibrary::NodeAddMenu(FParseRecord& Record)
{
    GET_CAST_ACTIVE_TAB_RETURN(ActiveGraphEditor, SGraphEditor, TSharedPtr<IMenu>())

    SGraphPanel* GraphPanel = ActiveGraphEditor->GetGraphPanel();

    FVector2D SpawnLocation;
    {
        TSharedPtr<SWindow> TopLevelWindow = FSlateApplication::Get().GetActiveTopLevelRegularWindow();

        if (TopLevelWindow.IsValid())
        {
            SpawnLocation = TopLevelWindow->GetPositionInScreen();
            FVector2D WindowSize = TopLevelWindow->GetSizeInScreen();

            SpawnLocation.X += WindowSize.X / 5;
            SpawnLocation.Y += WindowSize.Y / 5;
        }
        else
        {
            FDisplayMetrics DisplayMetrics;
            FSlateApplication::Get().GetDisplayMetrics(DisplayMetrics);

            SpawnLocation = FVector2D(
                DisplayMetrics.PrimaryDisplayWidth / 5,
                DisplayMetrics.PrimaryDisplayHeight / 5
            );
        }

        TSharedPtr<SWidget> ContextWidgetToFocus = GraphPanel->SummonContextMenu(
            SpawnLocation,
            GraphPanel->GetPastePosition(),
            nullptr,
            nullptr,
            TArray<UEdGraphPin *>()
        );

        if (!ContextWidgetToFocus.IsValid())
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeAddMenu: Context Keyboard Focus Widget Not Found"));
            return TSharedPtr<IMenu>();
        }

        FWidgetPath KeyboardFocusPath;
        if (FSlateApplication::Get().FindPathToWidget(ContextWidgetToFocus.ToSharedRef(), KeyboardFocusPath))
        {
            return FSlateApplication::Get().FindMenuInWidgetPath(KeyboardFocusPath);
        }
        else
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeAddMenu: IMenu Could Not Be Found In Widget Path"))
            return TSharedPtr<IMenu>();
        }
    }
}

TSharedPtr<IMenu> UNodeInteractionLibrary::NodeAddPinMenu(FParseRecord &Record)
{
    GET_CAST_ACTIVE_TAB_RETURN(ActiveGraphEditor, SGraphEditor, TSharedPtr<IMenu>())

    SGraphPanel* GraphPanel = ActiveGraphEditor->GetGraphPanel();

    FVector2D SpawnLocation;
    {
        TSharedPtr<SWindow> TopLevelWindow = FSlateApplication::Get().GetActiveTopLevelRegularWindow();

        if (TopLevelWindow.IsValid())
        {
            SpawnLocation = TopLevelWindow->GetPositionInScreen();
            FVector2D WindowSize = TopLevelWindow->GetSizeInScreen();

            SpawnLocation.X += WindowSize.X / 5;
            SpawnLocation.Y += WindowSize.Y / 5;
        }
        else
        {
            FDisplayMetrics DisplayMetrics;
            FSlateApplication::Get().GetDisplayMetrics(DisplayMetrics);

            SpawnLocation = FVector2D(
                DisplayMetrics.PrimaryDisplayWidth / 5,
                DisplayMetrics.PrimaryDisplayHeight / 5
            );
        }

        TSharedRef<FGraphIndexer> Indexer =
    GetAssetRegistry()->GetGraphIndexer(ActiveGraphEditor->GetCurrentGraph());

        UParseIntInput* NodeIndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("NODE_INDEX"));
        UParseIntInput* PinIndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("PIN_INDEX"));

        if (NodeIndexInput == nullptr || PinIndexInput == nullptr)
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeAddMenu: Invalid Inputs"));

            return TSharedPtr<IMenu>();
        }

        TSharedPtr<SWidget> ContextWidgetToFocus = GraphPanel->SummonContextMenu(
            SpawnLocation,
            GraphPanel->GetPastePosition(),
            nullptr,
            nullptr,
            TArray<UEdGraphPin*> {
                Indexer->GetPin(
                    NodeIndexInput->GetValue(),
                    PinIndexInput->GetValue()
                )
            }
        );

        if (!ContextWidgetToFocus.IsValid())
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeAddMenu: Context Keyboard Focus
    Widget Not Found"));
            return TSharedPtr<IMenu>();
        }

        FWidgetPath KeyboardFocusPath;
        if (FSlateApplication::Get().FindPathToWidget(ContextWidgetToFocus.ToSharedRef(),
    KeyboardFocusPath))
        {
            return FSlateApplication::Get().FindMenuInWidgetPath(KeyboardFocusPath);
        }
        else
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeAddMenu: IMenu Could Not Be
    Found In Widget Path"))
            return TSharedPtr<IMenu>();
        }
    }
}

void UNodeInteractionLibrary::NodeAddSelect(FParseRecord& Record)
{
    GET_TOP_CONTEXT(Record, ContextMenu, UAccessibilityGraphEditorContext)

    UParseIntInput* IndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("SELECTION_INDEX"));
    if (IndexInput == nullptr)
        return;

    ContextMenu->SelectAction(IndexInput->GetValue());
}

void UNodeInteractionLibrary::NodeAddSearchAdd(FParseRecord& Record)
{
    GET_TOP_CONTEXT(Record, ContextMenu, UAccessibilityGraphEditorContext)

    UParseStringInput *SearchInput = Record.GetPhraseInput<UParseStringInput>(TEXT("SEARCH_PHRASE"));
    if (SearchInput == nullptr)
        return;

    ContextMenu->AppendFilterText(SearchInput->GetValue());
}

void UNodeInteractionLibrary::NodeAddSearchRemove(FParseRecord& Record)
```

```
{
    GET_TOP_CONTEXT(Record, ContextMenu, UAccessibilityGraphEditorContext);

    UParseIntInput* RemoveAmountInput = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (RemoveAmountInput == nullptr)
        return;

    ContextMenu->SetFilterText(
        EventUtils::RemoveWordsFromEnd(ContextMenu->GetFilterText(), RemoveAmountInput->GetValue())
    );
}

void UNodeInteractionLibrary::NodeAddSearchReset(FParseRecord& Record)
{
    GET_TOP_CONTEXT(Record, ContextMenu, UAccessibilityGraphEditorContext)

    ContextMenu->SetFilterText(TEXT(""));
}

void UNodeInteractionLibrary::NodeAddScroll(FParseRecord& Record)
{
    GET_TOP_CONTEXT(Record, ContextMenu, UAccessibilityGraphEditorContext)

    UParseEnumInput* DirectionInput = Record.GetPhraseInput<UParseEnumInput>(TEXT("DIRECTION"));
    UParseIntInput* AmountInput = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (DirectionInput == nullptr || AmountInput == nullptr)
        return;

    switch (EPhraseScrollInput(DirectionInput->GetValue()))
    {
        case EPhraseScrollInput::UP:
            ContextMenu->AppendScrollDistance(-AmountInput->GetValue());
            break;

        case EPhraseScrollInput::DOWN:
            ContextMenu->AppendScrollDistance(AmountInput->GetValue());
            break;

        case EPhraseScrollInput::TOP:
            ContextMenu->SetScrollDistanceTop();
            break;

        case EPhraseScrollInput::BOTTOM:
            ContextMenu->SetScrollDistanceBottom();
                break;

        default:
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("NodeAddScroll: Invalid Scroll
 Position / Direction"));
            return;
    }
}

void UNodeInteractionLibrary::SelectionNodeToggle(FParseRecord& Record)
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor);

    UParseIntInput* IndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("NODE_INDEX"));
    if (IndexInput == nullptr)
        return;

    TSharedRef<FGraphIndexer> Indexer = GetAssetRegistry()->GetGraphIndexer(
        ActiveGraphEditor->GetCurrentGraph()
    );

    UEdGraphNode* Node = Indexer->GetNode(IndexInput->GetValue());
    if (Node == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("SelectionToggle: Node Not Found"));
        return;
    }

    ActiveGraphEditor->SetNodeSelection(
        Node,
        !ActiveGraphEditor->GetSelectedNodes().Contains(Node)
    );
}

void UNodeInteractionLibrary::SelectionReset(FParseRecord &Record) {
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    ActiveGraphEditor->ClearSelectionSet();
}

void UNodeInteractionLibrary::SelectionMove(FParseRecord& Record)
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UParseEnumInput* Direction = Record.GetPhraseInput<UParseEnumInput>(TEXT("DIRECTION"));
    UParseIntInput* Amount = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (Direction == nullptr || Amount == nullptr)
        return;

    for (UObject* NodeObj : ActiveGraphEditor->GetSelectedNodes())
    {
        UEdGraphNode* Node = Cast<UEdGraphNode>(NodeObj);
        if (Node == nullptr)
            continue;

        switch (EPhrase2DDirectionalInput(Direction->GetValue()))
        {
            case EPhrase2DDirectionalInput::UP:
                Node->NodePosY -= Amount->GetValue();
                break;

            case EPhrase2DDirectionalInput::DOWN:
                Node->NodePosY += Amount->GetValue();
                break;

            case EPhrase2DDirectionalInput::LEFT:
                Node->NodePosX -= Amount->GetValue();
                break;

            case EPhrase2DDirectionalInput::RIGHT:
                Node->NodePosX += Amount->GetValue();
                break;

            default:
                UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("SelectionMove: Invalid
 Direction"));
                return;
        }
    }
}

void UNodeInteractionLibrary::SelectionAlignment(FParseRecord& Record)
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UParseEnumInput* PositionInput = Record.GetPhraseInput<UParseEnumInput>(TEXT("POSITION"));
    if (PositionInput == nullptr)
        return;

    switch (EPhrasePositionalInput(PositionInput->GetValue()))
    {
        case EPhrasePositionalInput::TOP:
            ActiveGraphEditor->OnAlignTop();
            break;

        case EPhrasePositionalInput::MIDDLE:
            ActiveGraphEditor->OnAlignMiddle();
            break;

        case EPhrasePositionalInput::BOTTOM:
            ActiveGraphEditor->OnAlignBottom();
            break;

        case EPhrasePositionalInput::LEFT:
            ActiveGraphEditor->OnAlignLeft();
            break;

        case EPhrasePositionalInput::RIGHT:
            ActiveGraphEditor->OnAlignRight();
            break;

        case EPhrasePositionalInput::CENTER:
            ActiveGraphEditor->OnAlignCenter();
            break;
    }
}

void UNodeInteractionLibrary::SelectionStraighten(FParseRecord& Record)
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    ActiveGraphEditor->OnStraightenConnections();
}

void UNodeInteractionLibrary::SelectionComment(FParseRecord& Record)
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UEdGraph* Graph = ActiveGraphEditor->GetCurrentGraph();

    TSharedPtr<FEdGraphSchemaAction> CommentCreateAction = Graph->GetSchema()->GetCreateCommentAction();
    if (CommentCreateAction.IsValid())
    {
        CommentCreateAction->PerformAction(Graph, nullptr, FVector2D(0, 0), true);
    }
    else UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("SelectionComment: Comment Creation Failed"));
}

void UNodeInteractionLibrary::LocomotionSelect(FParseRecord& Record)
{
    GET_TOP_CONTEXT(Record, LocomotionContext, UAccessibilityGraphLocomotionContext);

    UParseIntInput* ViewSelectionInput = Record.GetPhraseInput<UParseIntInput>(TEXT("INDEX"));
    if (ViewSelectionInput == nullptr)
        return;

    if (!LocomotionContext->SelectChunk(ViewSelectionInput->GetValue()))
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("Locomotion Select: Failed to Choose New View."));
    }
}

void UNodeInteractionLibrary::LocomotionRevert(FParseRecord& Record)
{
    GET_TOP_CONTEXT(Record, LocomotionContext, UAccessibilityGraphLocomotionContext);

    if (!LocomotionContext->RevertToPreviousView())
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("Locomotion Revert: Failed to Revert to Previous View."));
    }
}

void UNodeInteractionLibrary::LocomotionConfirm(FParseRecord& Record)
{
    GET_TOP_CONTEXT(Record, LocomotionContext, UAccessibilityGraphLocomotionContext);

    LocomotionContext->ConfirmSelection();
}

void UNodeInteractionLibrary::LocomotionCancel(FParseRecord& Record)
{
    GET_TOP_CONTEXT(Record, LocomotionContext, UAccessibilityGraphLocomotionContext);

    LocomotionContext->CancelLocomotion();
}

void UNodeInteractionLibrary::BlueprintCompile(FParseRecord& Record)
{
    GET_CAST_ACTIVE_TAB(ActiveGraphEditor, SGraphEditor)

    UEdGraph* ActiveGraph = ActiveGraphEditor->GetCurrentGraph();
    if (ActiveGraph == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("BlueprintCompile: Active Graph Not Found"));
        return;
    }

    UBlueprint* FoundBlueprint = FBlueprintEditorUtils::FindBlueprintForGraph(ActiveGraph);
    if (FoundBlueprint == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("BlueprintCompile: Blueprint Not Found"));
        return;
    }

    TSharedPtr<FBlueprintEditor> BlueprintEditor = StaticCastSharedPtr<FBlueprintEditor>(FKismetEditorUtilities::GetIBlueprintEditorForObject(FoundBlueprint, false));
    if (!BlueprintEditor.IsValid())
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Warning, TEXT("BlueprintCompile: BlueprintEditor Not Found"));
        return;
    }

    BlueprintEditor->Compile();
}
```

5.26 ViewInteractionLibrary.cpp

```cpp
#include "PhraseEvents/ViewInteractionLibrary.h"
#include "PhraseEvents/Utils.h"

#include "PhraseTree/Containers/Input/InputContainers.h"

#include "AssetAccessibilityRegistry.h"

#include "PhraseTree/PhraseInputNode.h"
#include "PhraseTree/PhraseDirectionalInputNode.h"
#include "PhraseTree/PhraseEventNode.h"

UViewInteractionLibrary::UViewInteractionLibrary(const FObjectInitializer &ObjectInitializer)
    : Super(ObjectInitializer)
{

}

UViewInteractionLibrary::~UViewInteractionLibrary()
{

}

void UViewInteractionLibrary::BindBranches(TSharedRef<FPhraseTree> PhraseTree)
{
    PhraseTree->BindBranch(
        MakeShared<FPhraseNode>(TEXT("VIEW"),
        TPhraseNodeArray {

            MakeShared<FPhraseNode>(TEXT("MOVE"),
            TPhraseNodeArray {

                MakeShared<FPhrase2DDirectionalInputNode>(TEXT("DIRECTION"),
                TPhraseNodeArray {

                    MakeShared<FPhraseInputNode<int32»(TEXT("AMOUNT"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
 &UViewInteractionLibrary::MoveViewport))

                    })

                })

            }),

            MakeShared<FPhraseNode>(TEXT("ZOOM"),
            TPhraseNodeArray {

                MakeShared<FPhrase2DDirectionalInputNode>(TEXT("DIRECTION"),
                TPhraseNodeArray {

                    MakeShared<FPhraseInputNode<int32»(TEXT("AMOUNT"),
                    TPhraseNodeArray {

                        MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
 &UViewInteractionLibrary::ZoomViewport))

                    })

                })

            }),

            MakeShared<FPhraseNode>(TEXT("FOCUS"),
            TPhraseNodeArray {

                MakeShared<FPhraseInputNode<int32»(TEXT("INDEX"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
 &UViewInteractionLibrary::IndexFocus))

                })

            })

        })
    );
}

void UViewInteractionLibrary::MoveViewport(FParseRecord &Record) {
    GET_ACTIVE_TAB(ActiveTab)

    FString TabType = ActiveTab->GetTypeAsString();

    UParseEnumInput* DirectionInput = Record.GetPhraseInput<UParseEnumInput>(TEXT("DIRECTION"));
    UParseIntInput* AmountInput = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (DirectionInput == nullptr || AmountInput == nullptr)
        return;

    if (TabType == "SGraphEditor")
    {
        TSharedPtr<SGraphEditor> GraphEditor = StaticCastSharedPtr<SGraphEditor>(ActiveTab);

        FVector2D ViewLocation;
        float ZoomAmount;
        GraphEditor->GetViewLocation(ViewLocation, ZoomAmount);

        switch (EPhrase2DDirectionalInput(DirectionInput->GetValue()))
        {
            case EPhrase2DDirectionalInput::UP:
                ViewLocation.Y -= AmountInput->GetValue();
                break;

            case EPhrase2DDirectionalInput::DOWN:
                ViewLocation.Y += AmountInput->GetValue();
                break;

            case EPhrase2DDirectionalInput::LEFT:
                ViewLocation.X -= AmountInput->GetValue();
                break;

            case EPhrase2DDirectionalInput::RIGHT:
                ViewLocation.X += AmountInput->GetValue();
                break;

            default:
                UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("MoveViewport: INVALID DIRECTION
INPUT"));
                return;
        }

        GraphEditor->SetViewLocation(ViewLocation, ZoomAmount);
    }

    // Further Viewport Implementation Here
}

void UViewInteractionLibrary::ZoomViewport(FParseRecord &Record)
{
    GET_ACTIVE_TAB(ActiveTab)

    FString TabType = ActiveTab->GetTypeAsString();

    UParseEnumInput* DirectionInput = Record.GetPhraseInput<UParseEnumInput>(TEXT("DIRECTION"));
    UParseIntInput* AmountInput = Record.GetPhraseInput<UParseIntInput>(TEXT("AMOUNT"));
    if (DirectionInput == nullptr || AmountInput == nullptr)
        return;

    if (TabType == "SGraphEditor")
    {
        TSharedPtr<SGraphEditor> GraphEditor = StaticCastSharedPtr<SGraphEditor>(ActiveTab);

        FVector2D ViewLocation;
        float ZoomAmount;
        GraphEditor->GetViewLocation(ViewLocation, ZoomAmount);

        switch (EPhrase2DDirectionalInput(DirectionInput->GetValue())) {
            case EPhrase2DDirectionalInput::UP:
                ZoomAmount += AmountInput->GetValue();
                break;

            case EPhrase2DDirectionalInput::DOWN:
                ZoomAmount -= AmountInput->GetValue();
                break;

            default:
                UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("ZoomViewport: INVALID DIRECTION
INPUT"));
                return;
        }

        GraphEditor->SetViewLocation(ViewLocation, ZoomAmount);
    }

    // Further Viewport Specific Implementation Here
}

void UViewInteractionLibrary::IndexFocus(FParseRecord& Record)
{
    GET_ACTIVE_TAB(ActiveTab)

    FString TabType = ActiveTab->GetTypeAsString();

    UParseIntInput* IndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("INDEX"));
    if (IndexInput == nullptr)
        return;

    if (TabType == "SGraphEditor")
    {
        TSharedPtr<SGraphEditor> GraphEditor = StaticCastSharedPtr<SGraphEditor>(ActiveTab);
        if (!GraphEditor.IsValid())
            return;

        TSharedRef<FAssetAccessibilityRegistry> AssetRegistry = GetAssetRegistry();

        TSharedRef<FGraphIndexer> GraphIndexer =
AssetRegistry->GetGraphIndexer(GraphEditor->GetCurrentGraph());

        UEdGraphNode* Node = GraphIndexer->GetNode(IndexInput->GetValue());
        if (Node == nullptr)
        {
            UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("IndexFocus: INVALID INDEX INPUT"))
            return;
        }

        GraphEditor->JumpToNode(Node);
    }

    // Further ViewportS Specific Implementation Here
}
```

5.27 WindowInteractionLibrary.cpp

```
#include "PhraseEvents/WindowInteractionLibrary.h"
#include "PhraseEvents/Utils.h"

#include "PhraseTree/PhraseInputNode.h"
#include "PhraseTree/PhraseEventNode.h"
#include "PhraseTree/Containers/Input/UParseIntInput.h"

#include "AccessibilityWrappers/AccessibilityWindowToolbar.h"

UWindowInteractionLibrary::UWindowInteractionLibrary(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer)
{
    WindowToolBar = NewObject<UAccessibilityWindowToolbar>();
}

UWindowInteractionLibrary::~UWindowInteractionLibrary()
{

}

void UWindowInteractionLibrary::BindBranches(TSharedRef<FPhraseTree> PhraseTree)
{
    PhraseTree->BindBranches(
        TPhraseNodeArray{

            MakeShared<FPhraseNode>(TEXT("WINDOW"),
            TPhraseNodeArray{

                MakeShared<FPhraseNode>(TEXT("CLOSE"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UWindowInteractionLibrary::CloseActiveWindow))

                }),

            }),

            MakeShared<FPhraseNode>(TEXT("TOOLBAR"),
            TPhraseNodeArray {

                MakeShared<FPhraseInputNode<int32»(TEXT("ITEM_INDEX"),
                TPhraseNodeArray {

                    MakeShared<FPhraseEventNode>(CreateParseDelegate(this,
&UWindowInteractionLibrary::SelectToolBarItem))

                })

            })

        }
    );
}

void UWindowInteractionLibrary::CloseActiveWindow(FParseRecord &Record) {
    FSlateApplication& SlateApp = FSlateApplication::Get();
    if (!SlateApp.CanDisplayWindows())
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("CloseActiveWindow: Slate Application
 cannot display windows."));
        return;
    }

    TSharedPtr<SWindow> ActiveWindow = SlateApp.GetActiveTopLevelWindow();
    if (!ActiveWindow.IsValid())
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("CloseActiveWindow: No Active Window
 Found."));
        return;
    }

    TSharedPtr<SWindow> RootWindow = FGlobalTabmanager::Get()->GetRootWindow();
    if (ActiveWindow->IsVisible() && ActiveWindow != RootWindow)
    {
        ActiveWindow->RequestDestroyWindow();
    }
}

void UWindowInteractionLibrary::SelectToolBarItem(FParseRecord& Record)
{
    UParseIntInput* ItemIndexInput = Record.GetPhraseInput<UParseIntInput>(TEXT("ITEM_INDEX"));
    if (ItemIndexInput == nullptr)
    {
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display, TEXT("SelectToolBarItem: No Item Index
 Found."));
        return;
    }

    WindowToolBar->SelectToolbarItem(ItemIndexInput->GetValue());
}
```

5.28 TranscriptionVisualizer.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "TranscriptionVisualizer.h"

#include "AccessibilityWidgets/SAccessibilityTranscriptionVis.h"

FTranscriptionVisualizer::FTranscriptionVisualizer()
{
    RegisterTicker();
}

FTranscriptionVisualizer::~FTranscriptionVisualizer()
{
    UnregisterTicker();
}

bool FTranscriptionVisualizer::Tick(float DeltaTime)
{
    if (VisWindow.IsValid())
    {
        UpdateVisualizer();
    }
    else if (FSlateApplication::Get().GetActiveTopLevelRegularWindow().IsValid() &&
 FSlateApplication::Get().IsActive())
    {
        ConstructVisualizer();
    }

    return true;
}

void FTranscriptionVisualizer::ConstructVisualizer()
{
    TSharedPtr<SAccessibilityTranscriptionVis> MenuContent = SNew(SAccessibilityTranscriptionVis)
        .VisAmount(2);

    MenuContent->ForceVolatile(true);

    FDisplayMetrics DisplayMetrics;
    FSlateApplication::Get().GetDisplayMetrics(DisplayMetrics);

    FVector2D VisPosition = FVector2D();

    if (FSlateApplication::Get().GetActiveTopLevelRegularWindow().IsValid())
    {
        VisPosition =
   FSlateApplication::Get().GetActiveTopLevelRegularWindow()->GetPositionInScreen();
    }
    VisPosition.X = DisplayMetrics.PrimaryDisplayWidth;
    VisPosition.Y = DisplayMetrics.PrimaryDisplayHeight;

    TSharedRef<SWindow> MenuWindow = SNew(SWindow)
        .Type(EWindowType::Normal)
        .SizingRule(ESizingRule::Autosized)
        .ScreenPosition(VisPosition)
        .ClientSize(FVector2D(10, 10))
        .IsPopupWindow(true)
        //.InitialOpacity(0.5f)
        .SupportsTransparency(EWindowTransparency::PerWindow)
        .ActivationPolicy(EWindowActivationPolicy::Always)
        .AdjustInitialSizeAndPositionForDPIScale(true)
        [
            MenuContent.ToSharedRef()
        ];

    TSharedPtr<SWindow> TopLevelWindow = FSlateApplication::Get().GetActiveTopLevelRegularWindow();

    MenuWindow->AssignParentWidget(TopLevelWindow);
    FSlateApplication::Get().AddWindowAsNativeChild(MenuWindow , TopLevelWindow.ToSharedRef(), true);

    VisWindow = MenuWindow.ToWeakPtr();
    VisContent = MenuContent.ToWeakPtr();
}

void FTranscriptionVisualizer::UpdateVisualizer()
{
    if (FSlateApplication::Get().IsActive())
    {
        VisWindow.Pin()->ShowWindow();

        // ReparentWindow();

        MoveVisualizer();
    }
    else VisWindow.Pin()->HideWindow();
}

void FTranscriptionVisualizer::ReparentWindow()
{
    TSharedPtr<SWindow> TopLevelActiveWindow =
   FSlateApplication::Get().GetActiveTopLevelRegularWindow();
    if (!TopLevelActiveWindow.IsValid())
        return;

    TSharedPtr<SWindow> VisWindowPtr = VisWindow.Pin();

    if (TopLevelActiveWindow == VisWindow.Pin() ||
        TopLevelActiveWindow->GetContent() == VisWindowPtr->GetParentWidget())
        return;

    TSharedPtr<SWindow> PrevParentWindow = VisWindowPtr->GetParentWindow();
    if (PrevParentWindow.IsValid())
    {
        PrevParentWindow->RemoveDescendantWindow(VisWindowPtr.ToSharedRef());
    }

    VisWindowPtr->AssignParentWidget(TopLevelActiveWindow);
    TopLevelActiveWindow->AddChildWindow(VisWindowPtr.ToSharedRef());
}

void FTranscriptionVisualizer::MoveVisualizer()
{
    FVector2D NewPosition = FVector2D();

    if (!GetTopScreenVisualizerPosition(NewPosition))
    {
        GetDisplayVisualizerPosition(NewPosition);
    }

    VisWindow.Pin()->MoveWindowTo(NewPosition);
}

void FTranscriptionVisualizer::OnTranscriptionRecieved(TArray<FString> InTranscription)
{
    for (int i = 0; i < InTranscription.Num(); i++)
    {
        VisContent.Pin()->UpdateTopTranscription(InTranscription[i]);
    }
}

bool FTranscriptionVisualizer::GetTopScreenVisualizerPosition(FVector2D& OutPosition)
{
    TSharedPtr<SWindow> TopLevelWindow = FSlateApplication::Get().GetActiveTopLevelRegularWindow();
    if (!TopLevelWindow.IsValid())
        return false;

    FVector2D ActiveWindowPosition = TopLevelWindow->GetPositionInScreen();
    FVector2D ActiveWindowBounds = TopLevelWindow->GetClientSizeInScreen();

    TSharedPtr<SWindow> VisWindowPtr = VisWindow.Pin();

    OutPosition.X = (ActiveWindowPosition.X + ActiveWindowBounds.X / 2) -
    (VisWindowPtr->GetClientSizeInScreen().X / 2);
    OutPosition.Y = (ActiveWindowPosition.Y + ActiveWindowBounds.Y - 50) -
   VisWindowPtr->GetClientSizeInScreen().Y;

    return true;
}

bool FTranscriptionVisualizer::GetDisplayVisualizerPosition(FVector2D& OutPosition)
{
    FDisplayMetrics DisplayMetrics;
    FSlateApplication::Get().GetDisplayMetrics(DisplayMetrics);

    OutPosition.X = DisplayMetrics.PrimaryDisplayWidth;
    OutPosition.Y = DisplayMetrics.PrimaryDisplayHeight;

    return true;
}

void FTranscriptionVisualizer::RegisterTicker()
{
    FTickerDelegate TickDelegate = FTickerDelegate::CreateRaw(this, &FTranscriptionVisualizer::Tick);

    TickDelegateHandle = FTSTicker::GetCoreTicker().AddTicker(TickDelegate);
}

void FTranscriptionVisualizer::UnregisterTicker()
{
    FTSTicker::GetCoreTicker().RemoveTicker(TickDelegateHandle);
}
```

5.29 WidgetUtils.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

template<class T>
[[nodiscard]] FORCEINLINE TSharedPtr<T> GetWidgetDescendant(const TSharedRef<SWidget>& SearchRoot,
   FString TargetWidgetType)
{
    static_assert(TIsDerivedFrom<T, SWidget>::IsDerived, "Provided Type Is Not a Valid Widget Type.");

    TargetWidgetType.RemoveSpacesInline();

    if (SearchRoot->GetType() == TargetWidgetType)
        return StaticCastSharedRef<T>(SearchRoot);

    {
        TArray<FChildren*> ChildrenToSearch = TArray{
            SearchRoot->GetChildren()
        };

        FChildren* CurrentChildren;
        TSharedPtr<SWidget> CurrentChild;
        FString CurrentChildString;

        while (ChildrenToSearch.Num() > 0)
        {
            CurrentChildren = ChildrenToSearch.Pop();

            for (int i = 0; i < CurrentChildren->Num(); i++)
            {
                CurrentChild = CurrentChildren->GetChildAt(i);

                CurrentChildString = CurrentChild->GetTypeAsString();
                CurrentChildString.RemoveSpacesInline();

                if (CurrentChildString == TargetWidgetType)
                    return StaticCastSharedPtr<T>(CurrentChild);

                ChildrenToSearch.Add(CurrentChild->GetChildren());
            }
        }
    }

    return TSharedPtr<T>();
}


template <class T>
[[nodiscard]] FORCEINLINE TArray<TSharedPtr<T» GetWidgetDescendants(const TSharedRef<SWidget>&
 SearchRoot, FString TargetWidgetType)
{
    static_assert(TIsDerivedFrom<T, SWidget>::IsDerived, "Provided Type Is Not a Valid Widget Type.");

    TargetWidgetType.RemoveSpacesInline();

    TArray<TSharedPtr<T» FoundDescendants = TArray<TSharedPtr<T»();

    if (SearchRoot->GetTypeAsString() == TargetWidgetType)
        FoundDescendants.Add(StaticCastSharedRef<T>(SearchRoot));

    {
        TArray<FChildren*> ChildrenToSearch = TArray {
            SearchRoot->GetChildren()
        };

        while (ChildrenToSearch.Num() > 0)
        {
            FChildren* CurrentChildren = ChildrenToSearch.Pop();

            for (int i = 0; i < CurrentChildren->Num(); i++)
            {
                TSharedPtr<SWidget> CurrentChild = CurrentChildren->GetChildAt(i);

                FString CurrentChildString = CurrentChild->GetTypeAsString();
                CurrentChildString.RemoveSpacesInline();

                if (CurrentChildString == TargetWidgetType)
                    FoundDescendants.Add(StaticCastSharedPtr<T>(CurrentChild));

                ChildrenToSearch.Add(CurrentChild->GetChildren());
            }
        }
    }

    return FoundDescendants;
}

[[nodiscard]] FORCEINLINE TArray<FSlotBase*> GetWidgetSlotsByType(const TSharedRef<SWidget>&
 SearchRoot, const TSet<FString>& TargetTypes)
{
    TArray<FSlotBase*> FoundDescendants = TArray<FSlotBase*>();

    {
        TArray<FChildren*> ChildrenToSearch = TArray{
            SearchRoot->GetChildren()
        };

        FChildren* CurrentChildren;
        FString CurrentWidgetString;

        while (ChildrenToSearch.Num() > 0)
        {
            CurrentChildren = ChildrenToSearch.Pop();

            for (int i = 0; i < CurrentChildren->NumSlot(); i++)
            {
                FSlotBase& CurrentSlot = const_cast<FSlotBase&>(CurrentChildren->GetSlotAt(i));

                const TSharedRef<SWidget> CurrentWidget = CurrentSlot.GetWidget();

                CurrentWidgetString = CurrentWidget->GetTypeAsString();
                CurrentWidgetString.RemoveSpacesInline();

                if (TargetTypes.Contains(CurrentWidgetString))
                    FoundDescendants.Add(&CurrentSlot);

                ChildrenToSearch.Add(CurrentWidget->GetChildren());
            }
        }
    }
    return FoundDescendants;
}
```

5.30 AccessibilityNodeFactory.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "NodeFactory.h"
#include "OpenAccessibility.h"
#include "AccessibilityWidgets/SIndexer.h"

#include "SGraphNode.h"
#include "SGraphPin.h"

template<class T>
class OPENACCESSIBILITY_API TGraphAccessibilityNodeFactory : public FGraphNodeFactory
{
public:

    static_assert(TIsDerivedFrom<T, FGraphNodeFactory>::IsDerived, "Provided Template Type Must Derive
 From FGraphNodeFactory");

    TGraphAccessibilityNodeFactory()
    {
        Implementation = TSharedPtr<T>();

        AccessibilityRegistry =
 FOpenAccessibilityModule::Get().AssetAccessibilityRegistry.ToSharedRef();
    }

    TGraphAccessibilityNodeFactory(TSharedRef<FAssetAccessibilityRegistry> InAccessibilityRegistry)
        : AccessibilityRegistry(InAccessibilityRegistry)
    {
        Implementation = TSharedPtr<T>();
    }

    virtual ~TGraphAccessibilityNodeFactory()
    {

    }

    /* FGraphNodeFactory Implementation */

    virtual TSharedPtr<class SGraphNode> CreateNodeWidget(UEdGraphNode* InNode) override;

    virtual TSharedPtr<class SGraphPin> CreatePinWidget(UEdGraphPin* InPin) override;

    /* End Of FGraphNodeFactory Implementation*/

protected:

    TSharedRef<FAssetAccessibilityRegistry> AccessibilityRegistry;

    TSharedPtr<T> Implementation;
};

template<class T>
TSharedPtr<class SGraphNode> TGraphAccessibilityNodeFactory<T>::CreateNodeWidget(UEdGraphNode* InNode)
{
    check(InNode != nullptr);

    TSharedPtr<SGraphNode> OutNode = Implementation->CreateNodeWidget(InNode);

    // Apply Accessibility Visuals to the Node.

    TSharedRef<FGraphIndexer> GraphIndexer =
 AccessibilityRegistry->GetGraphIndexer(InNode->GetGraph());

    int NodeIndex = -1;
    GraphIndexer->GetOrAddNode(InNode);

    TSharedRef<SWidget> WidgetToWrap = OutNode->GetSlot(ENodeZone::Center)->GetWidget();

    check(WidgetToWrap != SNullWidget::NullWidget);

    OutNode->GetOrAddSlot(ENodeZone::Center)
        .HAlign(HAlign_Fill)
        [
            SNew(SVerticalBox)
```

```
                + SVerticalBox::Slot()
                .HAlign(HAlign_Fill)
                .AutoHeight()
                .Padding(FMargin(1.5f, 0.25f))
                [
                    SNew(SOverlay)

                        + SOverlay::Slot()
                        [
                            SNew(SImage)
                                .Image(FAppStyle::Get().GetBrush("Graph.Node.Body"))
                        ]

                        + SOverlay::Slot()
                        .Padding(FMargin(4.0f, 0.0f))
                        [
                            SNew(SHorizontalBox)
                                + SHorizontalBox::Slot()
                                .HAlign(HAlign_Right)
                                .VAlign(VAlign_Center)
                                .Padding(1.f)
                                [
                                    SNew(SOverlay)
                                        + SOverlay::Slot()
                                        [
                                            SNew(SIndexer)
                                                .IndexValue(NodeIndex)
                                                .TextColor(FLinearColor::White)
                                                .BorderColor(FLinearColor::Gray)
                                        ]
                                ]
                        ]
                ]

                + SVerticalBox::Slot()
                .HAlign(HAlign_Fill)
                .AutoHeight()
                [
                    WidgetToWrap
                ]
        ];

    return OutNode;
}

template<class T>
TSharedPtr<class SGraphPin> TGraphAccessibilityNodeFactory<T>::CreatePinWidget(UEdGraphPin* InPin)
{
    check(InPin != nullptr);

    TSharedPtr<SGraphPin> OutPin = Implementation->CreatePinWidget(InPin);
    SGraphPin* OutPinPtr = OutPin.Get();

    TSharedRef<FGraphIndexer> GraphIndexer =
 AccessibilityRegistry->GetGraphIndexer(InPin->GetOwningNode()->GetGraph());

    int PinIndex = -1;
    PinIndex = InPin->GetOwningNode()->GetPinIndex(InPin);

    TSharedRef<SWidget> AccessiblityWidget = SNew(SOverlay)
        .Visibility_Lambda([OutPinPtr]() -> EVisibility {
            if (OutPinPtr->HasAnyUserFocusOrFocusedDescendants() || OutPinPtr->IsHovered())
                return EVisibility::Visible;

            return EVisibility::Hidden;
        })
        + SOverlay::Slot()
        [
            SNew(STextBlock)
                .ColorAndOpacity(FLinearColor::White)
                .ShadowColorAndOpacity(FLinearColor::Black)
                .ShadowOffset(FIntPoint(-1, 1))
                .Font(FAppStyle::Get().GetFontStyle("Graph.Node.Pin.Font"))
                .Text(FText::FromString("[" + FString::FromInt(PinIndex) + "]"))
        ];

    // Get Pin Widget Content, before modifying it.
    TSharedRef<SWidget> PinWidgetContent = OutPin->GetContent();

    // Modify the Pin Widget Content, based on the Pin's Direction.
    switch (OutPin->GetDirection())
    {
    case EEdGraphPinDirection::EGPD_Input:
    {
        OutPin->SetContent(
            SNew(SHorizontalBox)
            + SHorizontalBox::Slot()
            [
                PinWidgetContent
            ]
            + SHorizontalBox::Slot()
            [
                AccessiblityWidget
            ]
        );

        break;
    }

    case EEdGraphPinDirection::EGPD_Output:
    {
        OutPin->SetContent(
            SNew(SHorizontalBox)
            + SHorizontalBox::Slot()
            .AutoWidth()
            [
                AccessiblityWidget
            ]
            + SHorizontalBox::Slot()
            .AutoWidth()
            [
                PinWidgetContent
            ]
        );

        break;
    }
    }

    return OutPin;
}
```

5.31 SAccessibilityTranscriptionVis.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "Styling/AppStyle.h"
#include "Widgets/Layout/SBorder.h"

class OPENACCESSIBILITY_API SAccessibilityTranscriptionVis : public SBox
{
public:

    SLATE_BEGIN_ARGS(SAccessibilityTranscriptionVis)
    : _VisAmount(1)
    {}
        SLATE_ARGUMENT( int, VisAmount )
    SLATE_END_ARGS()

    ~SAccessibilityTranscriptionVis();

    void Construct(const FArguments& InArgs);

    // SWidget Interface

    virtual void Tick(const FGeometry& AllottedGeometry, const double InCurrentTime, const float
 InDeltaTime) override;

    // End of SWidget Interface

    void UpdateTopTranscription(const FString& InTopTranscription);

protected:

    TWeakPtr<SVerticalBox> TranscriptionContainer;

    TArray<TWeakPtr<STextBlock>> TranscriptionSlots;

};
```

5.32 SContentIndexer.h

```
// Copyright F-Dudley. All Rights Reserved.
```

```

#pragma once

#include "CoreMinimal.h"
#include "Widgets/DeclarativeSyntaxSupport.h"

enum class EIndexerPosition : uint8
{
    Top,
    Bottom,
    Left,
    Right
};

class OPENACCESSIBILITY_API SContentIndexer : public SBox
{
public:

    SLATE_BEGIN_ARGS( SContentIndexer )
        : _IndexValue(0)
        , _IndexPositionToContent(EIndexerPosition::Left)
        , _ContentToIndex(SNullWidget::NullWidget)
        {}
        SLATE_ARGUMENT(int32, IndexValue)
        SLATE_ARGUMENT(EIndexerPosition, IndexPositionToContent)
        SLATE_ARGUMENT(TSharedPtr<SWidget>, ContentToIndex)

        SLATE_PRIVATE_ATTRIBUTE_VARIABLE(EVisibility, IndexVisibility) = EVisibility::Visible;
        SLATE_PRIVATE_ATTRIBUTE_FUNCTION(EVisibility, IndexVisibility)
    SLATE_END_ARGS()

    ~SContentIndexer();


    void Construct(const FArguments& InArgs);

    // SWidget Implementation

    virtual void Tick(const FGeometry& AllottedGeometry, const double InCurrentTime, const float
 InDeltaTime) override;

    // End SWidget Implementation

    void UpdateIndex(const int32 IndexValue);

    TSharedRef<SWidget> GetContent() const
    {
        return IndexedContent.Pin().ToSharedRef();
    }

    template<typename CastType>
    TSharedRef<CastType> GetContent() const
    {
        return CastStaticSharedPtr<CastType>(IndexedContent.Pin());
    }

protected:

    TSharedPtr<SWidget> ConstructTopIndexer(const FArguments& InArgs);

    TSharedPtr<SWidget> ConstructBottomIndexer(const FArguments& InArgs);

    TSharedPtr<SWidget> ConstructLeftIndexer(const FArguments& InArgs);

    TSharedPtr<SWidget> ConstructRightIndexer(const FArguments& InArgs);

    TSharedPtr<SWidget> ConstructContentContainer(TSharedRef<SWidget> ContentToIndex);

    TSharedPtr<SWidget> ConstructIndexContainer(const FArguments& InArgs, FLinearColor TextColor =
 FLinearColor::White);

    FText ConstructIndexText(int32 Index);

protected:

    TWeakPtr<SWidget> IndexedContent;

    TWeakPtr<class SIndexer> IndexerWidget;
};
```

5.33 SIndexer.h

```
// Copyright F-Dudley. All Rights Reserved.
```

```

#pragma once

#include "CoreMinimal.h"

class OPENACCESSIBILITY_API SIndexer : public SBox {
public:

    SLATE_BEGIN_ARGS( SIndexer )
    : _TextColor(FLinearColor::White)
    , _BorderColor(FLinearColor::Black)
    {}
        SLATE_ARGUMENT(FLinearColor, TextColor)
        SLATE_ARGUMENT(FLinearColor, BorderColor)

        SLATE_PRIVATE_ARGUMENT_VARIABLE(int32, IndexValue) = -1;
        SLATE_PRIVATE_ARGUMENT_FUNCTION(int32, IndexValue)
        SLATE_PRIVATE_ATTRIBUTE_VARIABLE(EVisibility, IndexVisibility) = EVisibility::Visible;
        SLATE_PRIVATE_ATTRIBUTE_FUNCTION(EVisibility, IndexVisibility)
    SLATE_END_ARGS()

    ~SIndexer();

    // SWidget Implementation

    virtual void Tick(const FGeometry& AllotedGeometry, const double InCurrentTime, const float
  InDeltaTime) override;

    void Construct(const FArguments& InArgs);

    // End SWidget Implementation

    void UpdateIndex(const int32 NewIndex);

    void UpdateIndex(const FString& NewIndex);

    void UpdateIndex(const FText& NewIndex);

    TSharedPtr<STextBlock> GetIndexText() const
    {
        return IndexTextBlock.IsValid() ? IndexTextBlock.Pin() : TSharedPtr<STextBlock>();
    }

protected:

    TWeakPtr<STextBlock> IndexTextBlock;

};
```

5.34 AccessibilityAddNodeContextMenu.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "PhraseTree/Containers/ContextMenuObject.h"

#include "SGraphActionMenu.h"
#include "GraphActionNode.h"

#include "AccessibilityAddNodeContextMenu.generated.h"

struct FGraphActionNode;

UCLASS()
class OPENACCESSIBILITY_API UAccessibilityAddNodeContextMenu : public UPhraseTreeContextMenuObject
{
    GENERATED_BODY()

public:

    UAccessibilityAddNodeContextMenu();
    UAccessibilityAddNodeContextMenu(TSharedRef<IMenu> Menu);
    UAccessibilityAddNodeContextMenu(TSharedRef<IMenu> Menu, TSharedRef<SGraphActionMenu> GraphMenu);
    UAccessibilityAddNodeContextMenu(TSharedRef<IMenu> Menu, TSharedRef<SGraphActionMenu> GraphMenu,
  TSharedRef<STreeView<TSharedPtr<FGraphActionNode>>> TreeView);

    ~UAccessibilityAddNodeContextMenu();

    virtual void Init(TSharedRef<IMenu> InMenu, TSharedRef<FPhraseNode> InContextRoot) override;

    void Init(TSharedRef<IMenu> InMenu, TSharedRef<SGraphActionMenu> InGraphMenu,
TSharedRef<STreeView<TSharedPtr<FGraphActionNode»> InTreeView);

    // -- UAccessibilityContextMenu Implementation

    virtual void Init(TSharedRef<IMenu> InMenu) override;

    virtual bool Tick(float DeltaTime) override;

    virtual bool Close() override;

    virtual void ScaleMenu(const float ScaleFactor = 1.5f) override;

    // -- End UAccessibilityContextMenu Implementation

    bool DoesItemsRequireRefresh();

    void RefreshAccessibilityWidgets();

    // Utility Interactions
    // Useful for simplifying common interactions.

    void GetGraphActionFromIndex(const int32 InIndex, FGraphActionNode* OutGraphAction);

    FGraphActionNode* GetGraphActionFromIndex(const int32 InIndex);

    TSharedPtr<FGraphActionNode> GetGraphActionFromIndexSP(const int32 InIndex);

    void SelectGraphAction(const int32 InIndex);

    void PerformGraphAction(const int32 InIndex);

    FString GetFilterText();

    void SetFilterText(const FString& InFilterText);

    void AppendFilterText(const FString& InFilterText);

    void ResetFilterText();

    void SetScrollDistance(const float InScrollDistance);

    void AppendScrollDistance(const float InScrollDistance);

    void SetScrollDistanceTop();

    void SetScrollDistanceBottom();

    void ToggleContextAwareness();

protected:

    void ApplyAccessibilityWidget(TSharedRef<STableRow<TSharedPtr<FGraphActionNode»> ItemWidget);

    void UpdateAccessibilityWidget(TSharedRef<STableRow<TSharedPtr<FGraphActionNode»> ItemWidget);

public:

    // Menu Components

    TWeakPtr<SGraphActionMenu> GraphMenu;

    TWeakPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeView;

    TWeakPtr<SEditableTextBox> FilterTextBox;

    TWeakPtr<SCheckBox> ContextAwarenessCheckBox;

protected:

    FString PrevFilterString;
    int32 PrevNumItemsBeingObserved;
    int32 PrevNumGeneratedChildren;
    double PrevScrollDistance;
};
```

5.35 AccessibilityGraphEditorContext.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
```

```
#include "PhraseTree/Containers/ContextMenuObject.h"

#include "SGraphActionMenu.h"
#include "GraphActionNode.h"

#include "AccessibilityGraphEditorContext.generated.h"

class SContentIndexer;

UCLASS()
class OPENACCESSIBILITY_API UAccessibilityGraphEditorContext : public UPhraseTreeContextMenuObject
{
    GENERATED_BODY()

public:

    UAccessibilityGraphEditorContext();

    // -- UPhraseTreeContextMenuObject Implementation


    virtual void Init(TSharedRef<IMenu> InMenu, TSharedRef<FPhraseNode> InContextRoot) override;

    virtual bool Tick(float DeltaTime) override;

    virtual bool Close() override;

    virtual void ScaleMenu(const float ScaleFactor = 1.5f) override;

    // -- End of UPhraseTreeContextMenuObject Implementation



    // -- Event Actions

    TSharedPtr<FGraphActionNode> GetTreeViewAction(const int32& InIndex);

    void SelectAction(const int32& InIndex);

    FString GetFilterText();

    void SetFilterText(const FString& NewString);

    void AppendFilterText(const FString& StringToAdd);

    void SetScrollDistance(const float NewDistance);

    void AppendScrollDistance(const float DistanceToAdd);

    void SetScrollDistanceTop();

    void SetScrollDistanceBottom();

protected:

    // Index Utils

    const int32 GetStaticIndexOffset();

    // Component Finders

    bool FindGraphActionMenu(const TSharedRef<SWidget>& SearchRoot);

    bool FindTreeView(const TSharedRef<SWidget>& SearchRoot);

    bool FindStaticComponents(const TSharedRef<SWidget>& SearchRoot);

    // Component Tickers

    struct FTreeViewTickRequirements
    {
    public:

        FTreeViewTickRequirements()
            : PrevSearchText(FString())
            , PrevNumItemsBeingObserved(-1)
            , PrevNumGeneratedChildren(-1)
            , PrevScrollDistance(-1.00)
        { }

        FString PrevSearchText;
        int32 PrevNumItemsBeingObserved;
        int32 PrevNumGeneratedChildren;
        double PrevScrollDistance;
    };

    bool TreeViewCanTick();

    bool TreeViewRequiresTick();

    void TickTreeViewAccessibility();

    // Widget Utils

    void UpdateAccessibilityWidget(const TSharedRef<SContentIndexer>& ContextIndexer, const int32&
 NewIndex);

    const TSharedRef<SContentIndexer> CreateAccessibilityWrapper(const TSharedRef<SWidget>&
 ContentToWrap, const int32& Index);

protected:

    FTreeViewTickRequirements TreeViewTickRequirements;

    TWeakPtr<SGraphActionMenu> GraphMenu = TWeakPtr<SGraphActionMenu>();
    TWeakPtr<SEditableTextBox> FilterTextBox = TWeakPtr<SEditableTextBox>();

    TWeakPtr<STreeView<TSharedPtr<FGraphActionNode»> TreeView =
 TWeakPtr<STreeView<TSharedPtr<FGraphActionNode»>();

    TArray<TWeakPtr<SCheckBox» CheckBoxes = TArray<TWeakPtr<SCheckBox»();
};
```

5.36 AccessibilityGraphLocomotionContext.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "PhraseTree/Containers/ContextObject.h"

#include "Widgets/Layout/SUniformGridPanel.h"

#include "AccessibilityGraphLocomotionContext.generated.h"

USTRUCT()
struct FGraphLocomotionChunk
{
    GENERATED_BODY()

public:

    void SetChunkBounds(FVector2D InTopLeft, FVector2D InBottomRight)
    {
        TopLeft = InTopLeft;
        BottomRight = InBottomRight;
    }

    void GetChunkBounds(FVector2D& OutTopLeft, FVector2D& OutBottomRight) const
    {
        OutTopLeft = TopLeft;
        OutBottomRight = BottomRight;
    }

    FVector2D GetChunkTopLeft() const { return TopLeft; }

    FVector2D GetChunkBottomRight() const { return BottomRight; }

    void SetVisColor(const FLinearColor& NewColor) const
    {
        if (ChunkVisWidget.IsValid())
            ChunkVisWidget.Pin()->SetBorderBackgroundColor(NewColor);
    }

public:

    FVector2D TopLeft;

    FVector2D BottomRight;

    TWeakPtr<SBox> ChunkWidget;

    TWeakPtr<SBorder> ChunkVisWidget;

    TWeakPtr<class SIndexer> ChunkIndexer;

};

```

```
struct FPanelViewPosition
{
public:

    FPanelViewPosition()
        : TopLeft(FVector2D::ZeroVector)
        , BotRight(FVector2D::ZeroVector)
    { }

    FPanelViewPosition(FVector2D InTopLeft, FVector2D InBotRight)
        : TopLeft(InTopLeft)
        , BotRight(InBotRight)
    { }

    bool operator!=(const FVector2D& Other)
    {
        return TopLeft != Other || BotRight != Other;
    }

    bool operator!=(const FPanelViewPosition& Other)
    {
        return TopLeft != Other.TopLeft || BotRight != Other.BotRight;
    }

    FVector2D TopLeft;
    FVector2D BotRight;
};

UCLASS()
class OPENACCESSIBILITY_API UAccessibilityGraphLocomotionContext : public UPhraseTreeContextObject
{
    GENERATED_BODY()

public:

    UAccessibilityGraphLocomotionContext(const FObjectInitializer& ObjectInitializer);

    virtual ~UAccessibilityGraphLocomotionContext();

    void Init();
    void Init(TSharedRef<SGraphEditor> InGraphEditor);

    bool SelectChunk(const int32& Index);

    bool RevertToPreviousView();

    void ConfirmSelection();

    void CancelLocomotion();

    virtual bool Close() override;

protected:

    bool MoveViewport(const FVector2D& InTopLeft, const FVector2D& InBottomRight) const;

    bool MoveViewport(const FPanelViewPosition& NewViewPosition) const;

    // Visuals Methods

    void ChangeChunkVis(const int32& Index, const FLinearColor& NewColor = FLinearColor::Yellow);

    void CreateVisualGrid(const TSharedRef<SGraphEditor> InGraphEditor);

    void GenerateVisualChunks(const TSharedRef<SGraphEditor> InGraphEditor, FIntVector2
 InVisualChunkSize = FIntVector2(10));

    void CalculateVisualChunksBounds();

    void RemoveVisualGrid();

    void HideNativeVisuals();

    void UnHideNativeVisuals();

    void OnFocusChanged(const FFocusEvent& FocusEvent, const FWeakWidgetPath& OldFocusedWidgetPath,
 const TSharedPtr<SWidget>& OldFocusedWidget, const FWidgetPath& NewFocusedWidgetPath, const
 TSharedPtr<SWidget>& NewFocusedWidget);


    void BindFocusChangedEvent();

    void UnbindFocusChangedEvent();

protected:

    FVector2D StartViewPosition; float StartViewZoom;

    FPanelViewPosition CurrentViewPosition;
    TArray<FPanelViewPosition> PreviousPositions;

    // Chunking References

    TArray<FGraphLocomotionChunk> ChunkArray;

    FIntVector2 ChunkSize;


    // Container References

    TWeakPtr<SUniformGridPanel> GridContainer;

    TWeakPtr<SOverlay> GridParent;

    TWeakPtr<SGraphEditor> LinkedEditor;

private:

    FTimerHandle SelectionTimerHandle;

    TMap<SWidget*, EVisibility> NativeWidgetVisibility;

    FDelegateHandle FocusChangedHandle;
};
```

5.37 AccessibilityWindowToolbar.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "Indexers/Indexer.h"

#include "AccessibilityWindowToolbar.generated.h"

UCLASS()
class OPENACCESSIBILITY_API UAccessibilityWindowToolbar : public UObject
{
    GENERATED_BODY()

public:

    UAccessibilityWindowToolbar();

    virtual ~UAccessibilityWindowToolbar();

    bool Tick(float DeltaTime);

    // -- Parse Events --

    void SelectToolbarItem(int32 Index);

    // -- End of Parse Events --

    bool IsActiveToolbar(const TSharedRef<SWidget>& ToolkitWidget);

    TSharedPtr<SWidget> GetActiveToolkitWidget() const;

private:

    bool ApplyToolbarIndexing(TSharedRef<SWidget> ToolkitWidget, TSharedRef<SWindow> ToolkitWindow);

    // Widget Getters

    TSharedPtr<SBorder> GetWindowContentContainer(TSharedRef<SWindow> WindowToFindContainer);

    bool GetToolKitToolBar(TSharedRef<SWidget> ToolKitWidget, TSharedPtr<SWidget>& OutToolBar);

    void BindTicker();

    void UnbindTicker();

public:

private:

    TWeakPtr<SWindow> LastTopWindow;

    TWeakPtr<SBorder> LastToolkitParent;

    TWeakPtr<SWidget> LastToolkit;

    FIndexer<int32, SMultiBlockBaseWidget*> ToolbarIndex;

    FTSTicker::FDelegateHandle TickDelegateHandle;

    TArray<IConsoleCommand*> ConsoleCommands;

};
```

5.38 AssetAccessibilityRegistry.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "GraphIndexer.h"

class UBehaviorTree;

class OPENACCESSIBILITY_API FAssetAccessibilityRegistry
{
public:
    FAssetAccessibilityRegistry();
    ~FAssetAccessibilityRegistry();

    // Graph Indexing

    bool IsGraphAssetRegistered(const UEdGraph* InGraph) const;

    bool RegisterGraphAsset(const UEdGraph* InGraph);

    bool RegisterGraphAsset(const UEdGraph* InGraph, const TSharedRef<FGraphIndexer> InGraphIndexer);

    bool UnregisterGraphAsset(const UEdGraph* InGraph);

    TSharedRef<FGraphIndexer> GetGraphIndexer(const UEdGraph* InGraph) const {
        if (GraphAssetIndex.Contains(InGraph->GraphGuid))
            return GraphAssetIndex[InGraph->GraphGuid].ToSharedRef();

        return TSharedRef<FGraphIndexer>();
    }

    void GetAllGraphKeyIndexes(TArray<FGuid>& OutGraphKeys) const;

    TArray<FGuid> GetAllGraphKeyIndexes() const;

    void GetAllGraphIndexes(TArray<TSharedPtr<FGraphIndexer»& OutGraphIndexes) const;

    TArray<TSharedPtr<FGraphIndexer» GetAllGraphIndexes();

    // Game World Indexing

    bool IsGameWorldAssetRegistered(const UWorld* InWorld) const;

    bool RegisterGameWorldAsset(const UWorld* InWorld);

    bool UnregisterGameWorldAsset(const UWorld* InWorld);

private:

    // Asset Register Events

    void OnAssetOpenedInEditor(UObject* OpenedAsset, IAssetEditorInstance* EditorInstance);

    void OnAssetEditorRequestClose(UObject* ClosingAsset, EAssetEditorCloseReason CloseReason);

    void EmptyGraphAssetIndex();

    void EmptyGameWorldAssetIndex();

    // Asset Editor Registers

    void RegisterBlueprintAsset(const UBlueprint* InBlueprint);

    void RegisterMaterialAsset(const UMaterial* InMaterial);

    void RegisterBehaviorTreeAsset(const UBehaviorTree* InBehaviorTree);

    void RegisterUWorldAsset(const UWorld* InWorld);

public:

    TMap<FGuid, TSharedPtr<FGraphIndexer>> GraphAssetIndex;

    //TMap<UWorld, FWorldIndexer*> GameWorldAssetIndex;

private:

    FDelegateHandle AssetOpenedInEditorHandle;
    FDelegateHandle AssetEditorRequestCloseHandle;
};
```

5.39 GraphIndexer.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

class UEdGraph;
class UEdGraphNode;
struct FEdGraphEditAction;

class OPENACCESSIBILITY_API FGraphIndexer
{
public:

    FGraphIndexer();
    FGraphIndexer(const UEdGraph* GraphToIndex);
    ~FGraphIndexer();

    bool ContainsKey(const int& InKey);

    int ContainsNode(UEdGraphNode* InNode);

    void ContainsNode(UEdGraphNode* InNode, int& OutIndex);

    int GetKey(const UEdGraphNode* InNode);

    bool GetKey(const UEdGraphNode* InNode, int& OutKey);

    void GetNode(const int& InIndex, UEdGraphNode* OutNode);

    UEdGraphNode* GetNode(const int& InIndex);

    void GetPin(const int& InNodeIndex, const int& InPinIndex, UEdGraphPin* OutPin);

    UEdGraphPin* GetPin(const int& InNodeIndex, const int& InPinIndex);

    int AddNode(const UEdGraphNode* Node);

    void AddNode(int& OutIndex, const UEdGraphNode& InNode);

    int GetOrAddNode(const UEdGraphNode* InNode);

    void GetOrAddNode(const UEdGraphNode* InNode, int& OutIndex);

    void RemoveNode(const int& InIndex);

    void RemoveNode(const UEdGraphNode* InNode);

    void OnGraphEvent(const FEdGraphEditAction& InAction);

    void OnGraphRebuild();

private:

    int GetAvailableIndex();

    void GetAvailableIndex(int& OutIndex);

    void BuildGraphIndex();

protected:

    TMap<int, UEdGraphNode*> IndexMap;

    TSet<int32> NodeSet;

    TQueue<int32> AvailableIndices;

    UEdGraph* LinkedGraph;

    FDelegateHandle OnGraphChangedHandle;
};
```

5.40 Indexer.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "OpenAccessibilityLogging.h"

template <typename KeyType, typename ValueType>
class FIndexer
{
public:

    FIndexer()
    {

    }

    virtual ~FIndexer()
    {

    }


    bool IsEmpty() const
    {
        return IndexMap.IsEmpty();
    }

    void Reset()
    {
        IndexMap.Reset();
        AvailableIndexes.Empty();
    }

    void Empty()
    {
        IndexMap.Empty();
        AvailableIndexes.Empty();
    }

    int32 Num() const
    {
        return IndexMap.Num();
    }

    void Num(int32& OutNum) const
    {
        OutNum = IndexMap.Num();
    }

    bool ContainsKey(const KeyType& InKey)
    {
        return IndexMap.Contains(InKey);
    }

    bool ContainsValue(const ValueType& InValue)
    {
        check(InValue != nullptr);

        const KeyType* FoundKey = IndexMap.FindKey(InValue);

        return FoundKey != nullptr;
    }

    const KeyType GetKey(const ValueType& InValue)
    {
        check(InValue != nullptr);

        return *IndexMap.FindKey(InValue);
    }

    bool GetKey(const ValueType& InValue, KeyType& OutKey)
    {
        check(InValue != nullptr);

        const KeyType* FoundKey = IndexMap.FindKey(InValue);

        if (FoundKey != nullptr)
        {
            OutKey = *FoundKey;

            return true;
        }
        else return false;
    }

    ValueType GetValue(const KeyType& InKey)
    {
        return *IndexMap.Find(InKey);
    }

    bool GetValue(const KeyType& InKey, ValueType& OutValue)
    {
        if (!IndexMap.Contains(InKey))
        {
            UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided Key is not recognised."));
            return false;
        }

        OutValue = *IndexMap.Find(InKey);

        return true;
    }

    KeyType AddValue(const ValueType& InValue)
    {
        check(InValue != nullptr);

        if (ContainsValue(InValue))
        {
            return GetKey(InValue);
        }

        KeyType NewKey;
        GetAvailableKey(NewKey);

        IndexMap.Add(NewKey, InValue);

        return NewKey;
    }

    void AddValue(const ValueType& InValue, KeyType& OutKey)
    {
        check(InValue != nullptr);

        if (ContainsValue(InValue))
        {
            OutKey = GetKey(InValue);
            return;
        }

        OutKey = GetAvailableKey();

        IndexMap.Add(OutKey, InValue);
    }

    KeyType GetKeyOrAddValue(const ValueType& InValue)
    {
        check(InValue != nullptr);

        KeyType FoundKey;
        if (GetKey(InValue, FoundKey))
            return FoundKey;

        return AddValue(InValue);
    }

    void GetKeyOrAddValue(const ValueType& InValue, KeyType& OutKey)
    {
        check(InValue != nullptr);

        if (GetKey(InValue, OutKey))
            return;

        OutKey = AddValue(InValue);
    }

    void RemoveValue(const KeyType& InKey)
    {
        if (!IndexMap.Contains(InKey))
        {
            UE_LOG(LogOpenAccessibility, Warning, TEXT("Provided Key Has No Pair in Index."));
            return;
        }

        IndexMap.Remove(InKey);
        AvailableIndexes.Enqueue(InKey);
    }

    void RemoveValue(const ValueType& InValue)
    {
        check(InValue != nullptr);

        KeyType FoundKey;
        if (GetKey(InValue, FoundKey))
        {
            IndexMap.Remove(FoundKey);
            AvailableIndexes.Enqueue(FoundKey);
        }
        else UE_LOG(LogOpenAccessibility, Log, TEXT("Provided Value Had No Associated Key."));
    }

protected:

    void GetAvailableKey(KeyType& OutKey)
    {
        if (!AvailableIndexes.IsEmpty() && AvailableIndexes.Dequeue(OutKey))
            return;

        OutKey = IndexMap.Num();
    }

    KeyType GetAvailableKey()
    {
        if (!AvailableIndexes.IsEmpty())
        {
            KeyType OutKey;
            if (AvailableIndexes.Dequeue(OutKey))
                return OutKey;
        }

        return IndexMap.Num();
    }

public:


protected:

    TMap<KeyType, ValueType> IndexMap;

    TQueue<KeyType> AvailableIndexes;
};
```

5.41 OAccessibilityNodeFactory.h

```
// Fill out your copyright notice in the Description page of Project Settings.

#pragma once

#include "CoreMinimal.h"
#include "EdGraphUtilities.h"

class OPENACCESSIBILITY_API FAccessibilityNodeFactory : public FGraphPanelNodeFactory
{

public:
    /* Begin FGraphPanelNodeFactory */
    virtual TSharedPtr<class SGraphNode> CreateNode(UEdGraphNode* Node) const override;
    /* End FGraphPanelNodeFactory */

public:
    FAccessibilityNodeFactory();
    ~FAccessibilityNodeFactory();

    inline void WrapNodeWidget(UEdGraphNode* Node, TSharedRef<SGraphNode> NodeWidget, int NodeIndex)
 const;

    inline void WrapPinWidget(UEdGraphPin* Pin, TSharedRef<SGraphPin> PinWidget, int PinIndex,
 SGraphNode* OwnerNode) const;

    void SetSharedPtr(TSharedPtr<FAccessibilityNodeFactory> InSharedPtr)
    {
        ThisPtr = InSharedPtr;
    }

private:

    TSharedPtr<FAccessibilityNodeFactory> ThisPtr;
};
```

5.42 OAEditorAccessibilityManager.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

class OPENACCESSIBILITY_API OAEditorAccessibilityManager
{
public:
    OAEditorAccessibilityManager();
    ~OAEditorAccessibilityManager();
};
```

5.43 OpenAccessibility.h

```
// Copyright Epic Games, Inc. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "Modules/ModuleManager.h"

#include "AssetAccessibilityRegistry.h"
#include "OAccessibilityNodeFactory.h"

class FOpenAccessibilityModule : public IModuleInterface
{

public:

    virtual void StartupModule() override;
    virtual void ShutdownModule() override;
    static FOpenAccessibilityModule& Get()
    {
        return FModuleManager::GetModuleChecked<FOpenAccessibilityModule>("OpenAccessibility");
    }

    virtual bool SupportsDynamicReloading() override
    {
        return false;
    }

private:

    // Phrase Branch Bindings

    void BindLocalizedInteractionBranch();

    void BindGraphInteractionBranch();

    void BindViewportInteractionBranch();

    // Transcription Visualization

    void CreateTranscriptionVisualization();

    void DestroyTranscriptionVisualization();

    // Console Commands

    void RegisterConsoleCommands();

    void UnregisterConsoleCommands();

public:

    // Accessibility Components

    TSharedPtr<class FAccessibilityNodeFactory> AccessibilityNodeFactory;

    TSharedPtr<class FAssetAccessibilityRegistry> AssetAccessibilityRegistry;

private:

    TSharedPtr<class FTranscriptionVisualizer> TranscriptionVisualizer;

    TArray<IConsoleCommand*> ConsoleCommands;
};
```

5.44 OpenAccessibilityLogging.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

OPENACCESSIBILITY_API DECLARE_LOG_CATEGORY_EXTERN(LogOpenAccessibility, Log, All);

DEFINE_LOG_CATEGORY(LogOpenAccessibility);
```

5.45 LocalizedInputLibrary.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "PhraseTree/PhraseTreeFunctionLibrary.h"

#include "LocalizedInputLibrary.generated.h"

UCLASS()
class ULocalizedInputLibrary : public UPhraseTreeFunctionLibrary
{
    GENERATED_BODY()

public:

    ULocalizedInputLibrary(const FObjectInitializer& ObjectInitializer);

    virtual ~ULocalizedInputLibrary();

    // UPhraseTreeFunctionLibrary Implementation

    virtual void BindBranches(TSharedRef<FPhraseTree> PhraseTree) override;

    // End of UPhraseTreeFunctionLibrary Implementation


    // Keyboard Input Implementation

    UFUNCTION()
    void KeyboardInputAdd(FParseRecord& Record);

    UFUNCTION()
    void KeyboardInputRemove(FParseRecord& Record);

    UFUNCTION()
    void KeyboardInputReset(FParseRecord& Record);

    UFUNCTION()
    void KeyboardInputConfirm(FParseRecord& Record);

    UFUNCTION()
    void KeyboardInputExit(FParseRecord& Record);

    // End of Keyboard Input Implementation


    // Mouse Input Implementation



    // End of Keyboard Input Implementation
};
```

5.46 NodeInteractionLibrary.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "PhraseTree/PhraseTreeFunctionLibrary.h"

#include "NodeInteractionLibrary.generated.h"

UCLASS()
class UNodeInteractionLibrary : public UPhraseTreeFunctionLibrary
{
    GENERATED_BODY()

public:

    UNodeInteractionLibrary(const FObjectInitializer& ObjectInitializer);

    virtual ~UNodeInteractionLibrary();

    // UPhraseTreeFunctionLibrary Implementation

    virtual void BindBranches(TSharedRef<FPhraseTree> PhraseTree) override;

    // End of UPhraseTreeFunctionLibrary Implementation


    // Node Implementation

    UFUNCTION()
    void MoveNode(FParseRecord& Record);

    UFUNCTION()
    void DeleteNode(FParseRecord& Record);

    UFUNCTION()
    void NodeIndexFocus(int32 Index);

    // End of Node Implementation


    // Pin Implementation

    UFUNCTION()
    void PinConnect(FParseRecord& Record);

    UFUNCTION()
    void PinDisconnect(FParseRecord& Record);

    // End of Pin Implementation


    // Node Add Implementation

    TSharedPtr<IMenu> NodeAddMenu(FParseRecord& Record);

    TSharedPtr<IMenu> NodeAddPinMenu(FParseRecord& Record);


    void NodeAddSelect(FParseRecord& Record);

    void NodeAddSearchAdd(FParseRecord& Record);

    void NodeAddSearchRemove(FParseRecord& Record);

    void NodeAddSearchReset(FParseRecord& Record);

    void NodeAddScroll(FParseRecord& Record);

    // End of Node Add Implementation


    // Selection Implementation

    UFUNCTION()
    void SelectionNodeToggle(FParseRecord& Record);

    UFUNCTION()
    void SelectionReset(FParseRecord &Record);

    UFUNCTION()
    void SelectionMove(FParseRecord &Record);

    UFUNCTION()
    void SelectionAlignment(FParseRecord &Record);

    UFUNCTION()
    void SelectionStraighten(FParseRecord &Record);

    UFUNCTION()
    void SelectionComment(FParseRecord &Record);

    // End of Selection Implementation


    // Locomotion Implementation

    UFUNCTION()
    void LocomotionSelect(FParseRecord& Record);

    UFUNCTION()
    void LocomotionRevert(FParseRecord& Record);

    UFUNCTION()
    void LocomotionConfirm(FParseRecord& Record);

    UFUNCTION()
    void LocomotionCancel(FParseRecord& Record);

    // End of Locomotion Implementations


    // Blueprint Specifics

    UFUNCTION()
    void BlueprintCompile(FParseRecord& Record);

    // End of Blueprint Specifics
};
```

5.47 ViewInteractionLibrary.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "PhraseTree/PhraseTreeFunctionLibrary.h"

#include "ViewInteractionLibrary.generated.h"

UCLASS()
class UViewInteractionLibrary : public UPhraseTreeFunctionLibrary
{
    GENERATED_BODY()

public:

    UViewInteractionLibrary(const FObjectInitializer& ObjectInitializer);

    virtual ~UViewInteractionLibrary();

    // UPhraseTreeFunctionLibrary Implementation

    void BindBranches(TSharedRef<FPhraseTree> PhraseTree) override;

    // End of UPhraseTreeFunctionLibrary Implementation


    void MoveViewport(FParseRecord& Record);

    void ZoomViewport(FParseRecord& Record);

    void IndexFocus(FParseRecord& Record);
};
```

5.48 WindowInteractionLibrary.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

```

```
#include "CoreMinimal.h"

#include "PhraseTree/PhraseTreeFunctionLibrary.h"

#include "WindowInteractionLibrary.generated.h"

UCLASS()
class UWindowInteractionLibrary : public UPhraseTreeFunctionLibrary
{
    GENERATED_BODY()

public:

    UWindowInteractionLibrary(const FObjectInitializer& ObjectInitializer);

    virtual ~UWindowInteractionLibrary();

    // UPhraseTreeFunctionLibrary Implementation

    void BindBranches(TSharedRef<FPhraseTree> PhraseTree) override;

    // End of UPhraseTreeFunctionLibrary Implementation


    // Window Interaction

    void CloseActiveWindow(FParseRecord& Record);

    // End Window Interaction


    // Window ToolBar Interaction

    void SelectToolBarItem(FParseRecord& Record);

    // End Window ToolBar Interaction


protected:

    UPROPERTY(BlueprintReadOnly)
    class UAccessibilityWindowToolbar* WindowToolBar;

};
```

5.49 TranscriptionVisualizer.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

class OPENACCESSIBILITY_API FTranscriptionVisualizer
{
public:

    FTranscriptionVisualizer();
    ~FTranscriptionVisualizer();

    virtual bool Tick(float DeltaTime);

    // Visualizer Methods

    void ConstructVisualizer();

    void UpdateVisualizer();

    void ReparentWindow();

    void MoveVisualizer();

    void OnTranscriptionRecieved(TArray<FString> InTranscription);

protected:

    bool GetTopScreenVisualizerPosition(FVector2D& OutPosition);

    bool GetDisplayVisualizerPosition(FVector2D& OutPosition);

    // Ticker Manager Methods

    void RegisterTicker();
    void UnregisterTicker();


protected:

    // Ticker Vars

    FTSTicker::FDelegateHandle TickDelegateHandle;

    // Vis Components

    TWeakPtr<SWindow> VisWindow;

    TWeakPtr<class SAccessibilityTranscriptionVis> VisContent;
};
```

5.50 OpenAccessibilityAnalytics.Build.cs

```
// Copyright Epic Games, Inc. All Rights Reserved.

using System.IO;
using UnrealBuildTool;

public class OpenAccessibilityAnalytics : ModuleRules
{
    public OpenAccessibilityAnalytics(ReadOnlyTargetRules Target) : base(Target)
    {
        PCHUsage = ModuleRules.PCHUsageMode.UseExplicitOrSharedPCHs;

        PublicIncludePaths.AddRange(
            new string[] {
                // ... add public include paths required here ...
            }
            );

        PrivateIncludePaths.AddRange(
            new string[] {
                // ... add other private include paths required here ...
            }
            );


        PublicDependencyModuleNames.AddRange(
            new string[]
            {
                "Core",
                // ... add other public dependencies that you statically link with here ...
            }
            );


        PrivateDependencyModuleNames.AddRange(
            new string[]
            {
                "Engine",
            }
            );


        DynamicallyLoadedModuleNames.AddRange(
            new string[]
            {
                // ... add any modules that your module loads dynamically here ...
            }
            );

        CircularlyReferencedDependentModules.AddRange(
            new string[]
            {

            }
        );
    }
}
```

5.51 OpenAccessibilityAnalytics.cpp

```
#include "OpenAccessibilityAnalytics.h"
```

```
#include "OpenAccessibilityAnalyticsLogging.h"

#include "HAL/PlatformFileManager.h"
#include "Misc/DateTime.h"

#define LOCTEXT_NAMESPACE "FOpenAccessibilityAnalyticsModule"

void FOpenAccessibilityAnalyticsModule::StartupModule()
{
    SessionBufferFile = GenerateFileForSessionLog();

    EnableDumpTick();
    AddConsoleCommands();
}

void FOpenAccessibilityAnalyticsModule::ShutdownModule()
{
    DisableDumpTick();
    RemoveConsoleCommands();
}

bool FOpenAccessibilityAnalyticsModule::DumpTick(float DeltaTime)
{
    if (EventBuffer.IsEmpty())
        return true;

    if (SessionBufferFile.IsEmpty())
        SessionBufferFile = GenerateFileForSessionLog();

    UE_LOG(LogOpenAccessibilityAnalytics, Log, TEXT("Dumping Event Log To File."));

    if (!WriteBufferToFile())
    {
        UE_LOG(LogOpenAccessibilityAnalytics, Warning, TEXT("EventLog Dumping Failed."));
    }

    return true;
}

FString FOpenAccessibilityAnalyticsModule::GenerateFileForSessionLog()
{
    FDateTime CurrentDateTime = FDateTime::Now();

    FString CombinedFileName = TEXT("[") + CurrentDateTime.ToString() + TEXT("] OA Event Log.log");
    return FPaths::ConvertRelativePathToFull(FPaths::ProjectSavedDir() +
 TEXT("Logs/OpenAccessibility/") + CombinedFileName);
}

bool FOpenAccessibilityAnalyticsModule::WriteBufferToFile()
{
    if (EventBuffer.IsEmpty())
        return false;

    FString CombindedString = FString("");
    LoggedEvent CurrEvent;
    while (!EventBuffer.IsEmpty())
    {
        CurrEvent = EventBuffer[0];
        EventBuffer.RemoveAt(0);

        CombindedString += FString::Printf(TEXT("| %s | - %s\r\n"), *CurrEvent.Title,
 *CurrEvent.Body);
    }

    if (FFileHelper::SaveStringToFile(
            CombindedString,
            *SessionBufferFile,
            FFileHelper::EEncodingOptions::AutoDetect,
            &IFileManager::Get(),
            EFileWrite::FILEWRITE_Append
    ))
    {

        return true;
    }

    return false;
}

void FOpenAccessibilityAnalyticsModule::EnableDumpTick()
{
    const double DumpDelayCheck = 20.0f;

    FTickerDelegate TickDelegate = FTickerDelegate::CreateRaw(this,
 &FOpenAccessibilityAnalyticsModule::DumpTick);
    DumpTickHandle = FTSTicker::GetCoreTicker().AddTicker(TickDelegate, DumpDelayCheck);
}
```

```

void FOpenAccessibilityAnalyticsModule::DisableDumpTick()
{
    if (DumpTickHandle.IsValid())
        FTSTicker::GetCoreTicker().RemoveTicker(DumpTickHandle);
}

void FOpenAccessibilityAnalyticsModule::AddConsoleCommands()
{
    ConsoleCommands.Add(IConsoleManager::Get().RegisterConsoleCommand(
        TEXT("OpenAccessibilityAnalytics.Debug.Add_Mock_Event"),
        TEXT("Adds a MOCK Event to the Eventbuffer"),

        FConsoleCommandWithArgsDelegate::CreateLambda(
            [this](const TArray<FString>& Args) {

                if (Args.Num() < 2)
                    return;

                FString EventTitle = Args[0];
                FString EventBody;

                for (int i = 1; i < Args.Num(); i++)
                {
                    EventBody += Args[i] + TEXT(" ");
                }

                this->LogEvent(*EventTitle, *EventBody);
            }
        )
    ));

    ConsoleCommands.Add(IConsoleManager::Get().RegisterConsoleCommand(
        TEXT("OpenAccessibilityAnalytics.Debug.ForceLogDump"),
        TEXT("Forces a Dump of the Active To Log File."),

        FConsoleCommandDelegate::CreateLambda(
            [this]() {
                this->DumpTick(0.0f);
            }
        )
    ));
}

void FOpenAccessibilityAnalyticsModule::RemoveConsoleCommands()
{
    IConsoleCommand* ConsoleCommand = nullptr;
    while (ConsoleCommands.Num() > 0)
    {
        ConsoleCommand = ConsoleCommands.Pop();

        IConsoleManager::Get().UnregisterConsoleObject(ConsoleCommand);

        delete ConsoleCommand;
        ConsoleCommand = nullptr;
    }
}

#undef LOCTEXT_NAMESPACE

IMPLEMENT_MODULE(FOpenAccessibilityAnalyticsModule, OpenAccessibilityAnalytics)
```

5.52 OpenAccessibilityAnalyticsLogging.h

```
// Copyright Epic Games, Inc. All Rights Reserved.

#pragma once

DECLARE_LOG_CATEGORY_EXTERN(LogOpenAccessibilityAnalytics, Log, All);

DEFINE_LOG_CATEGORY(LogOpenAccessibilityAnalytics);
```

5.53 OpenAccessibilityAnalytics.h

```
// Copyright Epic Games, Inc. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
```

```
#include "Modules/ModuleManager.h"

#define OA_LOG(CategoryName, Verbosity, EventTitle, Format, ...) \
{ \
    UE_VALIDATE_FORMAT_STRING(Format, ##__VA_ARGS__); \
    UE_LOG(CategoryName, Verbosity, Format, ##__VA_ARGS__) \
    FOpenAccessibilityAnalyticsModule::Get().LogEvent(EventTitle, Format, ##__VA_ARGS__); \
}

class FOpenAccessibilityAnalyticsModule : public IModuleInterface {

public:

    virtual void StartupModule() override;
    virtual void ShutdownModule() override;

    virtual bool SupportsDynamicReloading() override { return false; }

    static FOpenAccessibilityAnalyticsModule& Get()
    {
        return FModuleManager::GetModuleChecked<FOpenAccessibilityAnalyticsModule>("OpenAccessibilityAnalytics");
    }

    bool DumpTick(float DeltaTime);

    // Analytics Logging

    void LogEvent(const TCHAR* EventTitle, const TCHAR* LogString, ...);

private:

    FString GenerateFileForSessionLog();

    bool WriteBufferToFile();

    void EnableDumpTick();

    void DisableDumpTick();

    void AddConsoleCommands();

    void RemoveConsoleCommands();

private:

    // Analytics Dumping

    FString SessionBufferFile;

    struct LoggedEvent
    {
    public:

        LoggedEvent()
        { };

        LoggedEvent(const TCHAR* EventTitle, const TCHAR* EventString, FDateTime EventTimestamp = FDateTime::Now())
            : Title(EventTitle)
            , Body(EventString)
            , Timestamp(EventTimestamp)
        { };

        LoggedEvent(const FString& EventTitle, const FString& EventString, FDateTime EventTimestamp = FDateTime::Now())
            : Title(EventTitle)
            , Body(EventString)
            , Timestamp(EventTimestamp)
        { };

    public:
        FString Title;
        FString Body;

        FDateTime Timestamp;
    };

    TArray<LoggedEvent> EventBuffer;

    FTSTicker::FDelegateHandle DumpTickHandle;

    // Console Commands

    TArray<IConsoleCommand*> ConsoleCommands;
};
```

```
FORCEINLINE void FOpenAccessibilityAnalyticsModule::LogEvent(const TCHAR* EventTitle, const TCHAR*
 LogString, ...)
{
    va_list Args;

    va_start(Args, LogString);
    TStringBuilder<1024> Message;
    Message.AppendV(LogString, Args);
    va_end(Args);

    EventBuffer.Add(
        LoggedEvent(EventTitle, *Message)
    );
}
```

5.54 OpenAccessibilityCommunication.Build.cs

```
// Copyright Epic Games, Inc. All Rights Reserved.

using System.IO;
using UnrealBuildTool;
using UnrealBuildTool.Rules;

public class OpenAccessibilityCommunication : ModuleRules
{
    public OpenAccessibilityCommunication(ReadOnlyTargetRules Target) : base(Target)
    {
        PCHUsage = ModuleRules.PCHUsageMode.UseExplicitOrSharedPCHs;

        PublicIncludePaths.AddRange(
            new string[] {
                // ... add public include paths required here ...
            }
            );

        PrivateIncludePaths.AddRange(
            new string[] {
                // ... add other private include paths required here ...
            }
            );


        PublicDependencyModuleNames.AddRange(
            new string[]
            {
                "Core",
                // ... add other public dependencies that you statically link with here ...
            }
            );

        PrivateDependencyModuleNames.AddRange(
            new string[]
            {
                // Internal Plugin Dependencies
                "OpenAccessibilityAnalytics",

                // Internal ThirdParty Dependencies
                "ZeroMQ",

                // Core Modules
                "CoreUObject",
                "Engine",
                "Json",

                // Editor Modules
                "UnrealEd",
                "Projects",

                // Slate UI Modules
                "Slate",
                "SlateCore",

                // Audio Modules
                "AudioMixer",
                "AudioCaptureCore",
                "AudioCapture",
                "InputCore",
            }
            );


        DynamicallyLoadedModuleNames.AddRange(
            new string[]
            {
                // ... add any modules that your module loads dynamically here ...
            }
            );

        CircularlyReferencedDependentModules.AddRange(
            new string[]
            {

            }
        );
    }
}
```

5.55 AudioManager.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "AudioManager.h"
#include "OpenAccessibilityCommunication.h"
#include "OpenAccessibilityComLogging.h"
#include "SocketCommunicationServer.h"

#include "AudioCaptureCore.h"
#include "AudioDeviceNotificationSubsystem.h"
#include "Templates/Function.h"

UAudioManager::UAudioManager()
{
    Settings = FAudioManagerSettings();

    // Create Audio Capture Object and Initialize Audio Stream
    bIsCapturingAudio = false;
    AudioCapture = NewObject<UAudioCapture>();
    AudioCapture->OpenDefaultAudioStream();
    AudioCapture->StartCapturingAudio();

    RegisterAudioGenerator();

    // Create FileIO Objects
    FileWriter = new Audio::FSoundWavePCMWriter();
}

UAudioManager::~UAudioManager()
{
    UnregisterAudioGenerator();

    AudioCapture->StopCapturingAudio();
    AudioCapture->RemoveFromRoot();

    delete AudioCapture; AudioCapture = nullptr;
    delete FileWriter; FileWriter = nullptr;
}

void UAudioManager::StartCapturingAudio()
{
    AudioBuffer.Empty();

    bIsCapturingAudio = true;
}

void UAudioManager::StopCapturingAudio()
{
    bIsCapturingAudio = false;

    if (AudioBuffer.Num() == 0)
        return;

    SaveAudioBufferToWAV(Settings.SavePath);

    if (OnAudioReadyForTranscription.ExecuteIfBound(AudioBuffer))
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Executing Audio Ready For Transcription
 Delegate. ||"));
    }
    else
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| No Delegates Bound to Audio Ready For
 Transcription Delegate. ||"));
    }

    AudioBuffer.Empty();
}

void UAudioManager::PRIVATE_OnAudioGenerate(const float* InAudio, int32 NumSamples)
{
    if (bIsCapturingAudio == false)
        return;

    // Need to Check Samples are above threshold and ignore if their run length is too long.

    AudioBuffer.Append(InAudio, NumSamples);
}

void UAudioManager::SaveAudioBufferToWAV(const FString& FilePath)
{
    UE_LOG(LogOpenAccessibilityCom, Log, TEXT("Starting to Save Audio Buffer to WAV"));

    Audio::FSampleBuffer SampleBuffer = Audio::FSampleBuffer(AudioBuffer.GetData(), AudioBuffer.Num(),
      AudioCapture->GetNumChannels(), AudioCapture->GetSampleRate());

    FileWriter->BeginWriteToWavFile(SampleBuffer, Settings.SaveName, const_cast<FString&>(FilePath),
      []() {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("Audio Buffer Saved to WAV"));
    });
}

void UAudioManager::OnDefaultDeviceChanged(EAudioDeviceChangedRole ChangedRole, FString DeviceID)
{
    UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Default Device Changed || Role: %d || DeviceID: %s
      ||"), ChangedRole, *DeviceID);

    this->UnregisterAudioGenerator();
    this->RegisterAudioGenerator();
}

void UAudioManager::RegisterAudioGenerator()
{
    // Add Audio Generator Delegate to get audio data from stream,
    // and apply wrapper function due to wanting to reference class function.
    OnAudioGenerateHandle = AudioCapture->AddGeneratorDelegate(FOnAudioGenerate([this](const float*
      InAudio, int32 NumSamples) {
        if (this->IsCapturingAudio()) this->PRIVATE_OnAudioGenerate(InAudio, NumSamples);
    }));
}

void UAudioManager::UnregisterAudioGenerator()
{
    AudioCapture->RemoveGeneratorDelegate(OnAudioGenerateHandle);
}
```

5.56 OpenAccessibilityComLogging.cpp

```

#include "OpenAccessibilityComLogging.h"
```

5.57 OpenAccessibilityCommunication.cpp

```
// Copyright Epic Games, Inc. All Rights Reserved.

#include "OpenAccessibilityCommunication.h"
#include "OpenAccessibilityComLogging.h"

#include "OpenAccessibilityAnalytics.h"

#include "AudioManager.h"
#include "SocketCommunicationServer.h"

#include "PhraseTree/PhraseNode.h"
#include "PhraseTree/PhraseInputNode.h"
#include "PhraseTree/PhraseDirectionalInputNode.h"
#include "PhraseTree/PhraseEventNode.h"

#include "Containers/Ticker.h"
#include "Dom/JsonObject.h"
#include "Interfaces/IPluginManager.h"
#include "Sound/SampleBufferIO.h"
#include "HAL/PlatformProcess.h"

#define LOCTEXT_NAMESPACE "UOpenAccessibilityCommunicationModule"

```

```
void FOpenAccessibilityCommunicationModule::StartupModule()
{
    LoadZMQDLL();

    // This code will execute after your module is loaded into memory; the exact timing is specified
 in the .uplugin file per-module
    UE_LOG(LogOpenAccessibilityCom, Display, TEXT("OpenAccessibilityComModule::StartupModule()"));

    // Initialize AudioManager
    AudioManager = NewObject<UAudioManager>();
    AudioManager->AddToRoot();

    AudioManager->OnAudioReadyForTranscription
        .BindRaw(this, &FOpenAccessibilityCommunicationModule::TranscribeWaveForm);

    // Initialize Socket Server
    SocketServer = MakeShared<FSocketCommunicationServer>();

    // Build The Phrase Tree
    BuildPhraseTree();

    // Bind Tick Event
    TickDelegate = FTickerDelegate::CreateRaw(this, &FOpenAccessibilityCommunicationModule::Tick);
    TickDelegateHandle = FTSTicker::GetCoreTicker().AddTicker(TickDelegate);

    // Bind Input Events
    KeyDownEventHandle = FSlateApplication::Get().OnApplicationPreInputKeyDownListener().AddRaw(this,
 &FOpenAccessibilityCommunicationModule::HandleKeyDownEvent);

    // Register Console Commands
    RegisterConsoleCommands();
}

void FOpenAccessibilityCommunicationModule::ShutdownModule()
{
    // This function may be called during shutdown to clean up your module.  For modules that support
 dynamic reloading,
    // we call this function before unloading the module.
    UE_LOG(LogOpenAccessibilityCom, Display, TEXT("OpenAccessibilityComModule::ShutdownModule()"));

    AudioManager->RemoveFromRoot();
    PhraseTreeUtils->RemoveFromRoot();

    FSlateApplication::Get().OnApplicationPreInputKeyDownListener().Remove(KeyDownEventHandle);

    UnloadZMQDLL();

    UnregisterConsoleCommands();
}

bool FOpenAccessibilityCommunicationModule::Tick(const float DeltaTime)
{
    // Detect if any events are ready to be received.
    if (SocketServer->EventOccured())
    {
        TArray<FString> RecvStrings;
        TSharedPtr<FJsonObject> RecvMetadata;

        // Receive the Detected Event, with separate transcriptions and metadata.
        if (SocketServer->RecvStringMultipartWithMeta(RecvStrings, RecvMetadata))
        {
            OA_LOG(LogOpenAccessibilityCom, Log, TEXT("TRANSCRIPTION RECIEVED"), TEXT("Recieved
 Multipart - Message Count: %d"), RecvStrings.Num());

            // Send Received Transcriptions to any bound events.
            OnTranscriptionRecieved.Broadcast(RecvStrings);
        }
    }

    return true;
}

void FOpenAccessibilityCommunicationModule::HandleKeyDownEvent(const FKeyEvent& InKeyEvent)
{
    // If the Space Key is pressed, we will send a request to the Accessibility Server
    if (InKeyEvent.GetKey() == EKeys::SpaceBar)
    {
        if (InKeyEvent.IsShiftDown())
        {
            OA_LOG(LogOpenAccessibilityCom, Log, TEXT("AudioCapture Change"), TEXT("Stopping Audio
 Capture"));
            AudioManager->StopCapturingAudio();
        }
        else
        {
            OA_LOG(LogOpenAccessibilityCom, Log, TEXT("AudioCapture Change"), TEXT("Starting Audio
 Capture"));
```

```
            AudioManager->StartCapturingAudio();
        }
    }
}

void FOpenAccessibilityCommunicationModule::TranscribeWaveForm(const TArray<float>
 AudioBufferToTranscribe)
{
    if (AudioBufferToTranscribe.Num() == 0)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Transcription Ready || Audio Buffer is Empty
 ||"));
        return;
    }

    PrevAudioBuffer = TArray(AudioBufferToTranscribe);

    UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| WaveForm Transcription || Array Size: %d || Byte
 Size: %s ||"), AudioBufferToTranscribe.Num(), *FString::FromInt(AudioBufferToTranscribe.Num() *
 sizeof(float)));

    // Create Metadata of Audio Source.
    TSharedPtr<FJsonObject> AudioBufferMetadata = MakeShared<FJsonObject>();
    AudioBufferMetadata->SetNumberField(TEXT("sample_rate"),
 AudioManager->GetAudioCaptureSampleRate());
    AudioBufferMetadata->SetNumberField(TEXT("num_channels"),
 AudioManager->GetAudioCaptureNumChannels());

    bool bArrayMessageSent = SocketServer->SendArrayMessageWithMeta(AudioBufferToTranscribe,
 AudioBufferMetadata.ToSharedRef(), ComSendFlags::none);

    OA_LOG(LogOpenAccessibilityCom, Log, TEXT("TRANSCRIPTION SENT"), TEXT("{%s} Send Audiobuffer
 (float x %d / %d Hz / %d channels)"),
        bArrayMessageSent ? TEXT("Success") : TEXT("Failed"),
        AudioBufferToTranscribe.Num(), AudioManager->GetAudioCaptureSampleRate(),
 AudioManager->GetAudioCaptureNumChannels());
}

void FOpenAccessibilityCommunicationModule::BuildPhraseTree()
{
    // Initialize the Phrase Tree
    PhraseTree = MakeShared<FPhraseTree>();
    PhraseTreePhraseRecievedHandle = OnTranscriptionRecieved
        .AddRaw(PhraseTree.Get(), &FPhraseTree::ParseTranscription);

    PhraseTreeUtils = NewObject<UPhraseTreeUtils>();
    PhraseTreeUtils->SetPhraseTree(PhraseTree.ToSharedRef());
    PhraseTreeUtils->AddToRoot();
}

void FOpenAccessibilityCommunicationModule::RegisterConsoleCommands()
{
    // Audio Commands

    ConsoleCommands.Add(IConsoleManager::Get().RegisterConsoleCommand(
        TEXT("OpenAccessibilityCom.Debug.ShowAudioSampleRate"),
        TEXT("Logs the Number of Samples being captured, from user input."),

        FConsoleCommandDelegate::CreateLambda([this]() {
            UE_LOG(LogOpenAccessibilityCom, Display,
 TEXT("OpenAccessibilityCom.Debug.ShowAudioSampleRate | Sample Rate: %d"),
 this->AudioManager->GetAudioCaptureSampleRate());
        })
    ));

    ConsoleCommands.Add(IConsoleManager::Get().RegisterConsoleCommand(
        TEXT("OpenAccessibilityCom.Debug.ShowAudioNumChannels"),
        TEXT("Logs the Number of Audio Channels being captured, from user input."),

        FConsoleCommandDelegate::CreateLambda([this]() {
            UE_LOG(LogOpenAccessibilityCom, Display,
 TEXT("OpenAccessibilityCom.Debug.ShowAudioNumChannels | Num Channels: %d"),
 this->AudioManager->GetAudioCaptureNumChannels());
        })
    ));

    ConsoleCommands.Add(IConsoleManager::Get().RegisterConsoleCommand(
        TEXT("OpenAccessibilityCom.Debug.SendLastBuffer"),
        TEXT("Sends the last saved audio buffer to the transcription service."),

        FConsoleCommandDelegate::CreateLambda([this]() {
            UE_LOG(LogOpenAccessibilityCom, Display,
 TEXT("OpenAccessibilityCom.Debug.SendLastBuffer"));

            TranscribeWaveForm(PrevAudioBuffer);
        })
    ));
```

```


}

void FOpenAccessibilityCommunicationModule::UnregisterConsoleCommands()
{
    IConsoleCommand* ConsoleCommand = nullptr;
    while (ConsoleCommands.Num() > 0)
    {
        ConsoleCommand = ConsoleCommands.Pop();

        IConsoleManager::Get().UnregisterConsoleObject(ConsoleCommand);
    }
}

void FOpenAccessibilityCommunicationModule::LoadZMQDLL()
{
    FString BaseDir = IPluginManager::Get().FindPlugin("OpenAccessibility")->GetBaseDir();

    FString LibraryPath;
#if PLATFORM_WINDOWS
    #if UE_BUILD_DEBUG
    LibraryPath = FPaths::Combine(*BaseDir,
 TEXT("Binaries/ThirdParty/ZeroMQ/Win64/libzmq-mt-gd-4_3_5.dll"));
    #else
    LibraryPath = FPaths::Combine(*BaseDir,
 TEXT("Binaries/ThirdParty/ZeroMQ/Win64/libzmq-mt-4_3_5.dll"));
    #endif
#elif PLATFORM_LINUX
    LibraryPath = FPaths::Combine(*BaseDir,
 TEXT("Binaries/ThirdParty/ZeroMQ/Linux/libzmq-mt-4_3_5.so"))
#elif PLATFORM_MAC
    LibraryPath = FPaths::Combine(*BaseDir,
 TEXT("Source/ThirdParty/ZeroMQ/Mac/libzmq-mt-4_3_5.dylib"))
#endif

    ZMQDllHandle = !LibraryPath.IsEmpty() ? FPlatformProcess::GetDllHandle(*LibraryPath) : nullptr;

    if (ZMQDllHandle)
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| LoadZMQDLL || Successfully Loaded ZMQ DLL ||"));
    }
    else
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("|| LoadZMQDLL || Failed to Load ZMQ DLL ||"));
    }
}

void FOpenAccessibilityCommunicationModule::UnloadZMQDLL()
{
    FPlatformProcess::FreeDllHandle(ZMQDllHandle);
    ZMQDllHandle = nullptr;
}

#undef LOCTEXT_NAMESPACE

IMPLEMENT_MODULE(FOpenAccessibilityCommunicationModule, OpenAccessibility)
```

5.58 PhraseTree.cpp

```
// Copyright F-Dudley. All Rights Reserved.


#include "PhraseTree.h"
#include "PhraseTree/PhraseNode.h"
#include "Algo/Reverse.h"

#include "Logging/StructuredLog.h"
#include "OpenAccessibilityComLogging.h"
#include "OpenAccessibilityAnalytics.h"

FPhraseTree::FPhraseTree() : FPhraseNode(TEXT("ROOT_NODE"))
{
    ContextManager = FPhraseTreeContextManager();

    FTickerDelegate TickDelegate = FTickerDelegate::CreateRaw(this, &FPhraseTree::Tick);
    TickDelegateHandle = FTSTicker::GetCoreTicker().AddTicker(TickDelegate);
}

FPhraseTree::~FPhraseTree()
{
    FTSTicker::GetCoreTicker().RemoveTicker(TickDelegateHandle);
}
```

```cpp

bool FPhraseTree::Tick(float DeltaTime)
{
    // Filter InActive Context Objects out of the stack.
    ContextManager.FilterContextStack();

    return true;
}

void FPhraseTree::ParseTranscription(TArray<FString> InTranscriptionSegments)
{
    if (InTranscriptionSegments.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Phrase Tree || Provided Transcription is Empty ||"))
        return;
    }

    TArray<FString> SegmentWordArray = TArray<FString>();
    int SegmentCount = 0;

    // Loop over any Transcription Segments.
    for (FString& TranscriptionSegment : InTranscriptionSegments)
    {
        if (TranscriptionSegment.IsEmpty())
        {
            UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Phrase Tree || Transcription Segment is Empty ||"))
            continue;
        }

        // Filter the Transcription Segment, to remove any unwanted characters.
        TranscriptionSegment.TrimStartAndEndInline();
        TranscriptionSegment.ReplaceInline(TEXT("."), TEXT(""), ESearchCase::IgnoreCase);
        TranscriptionSegment.ReplaceInline(TEXT(","), TEXT(""), ESearchCase::IgnoreCase);
        TranscriptionSegment.ToUpperInline();

        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Phrase Tree || Filtered Transcription Segment: { %s } ||"), *TranscriptionSegment)

        // Parse the Transcription Segment into an Array of Words, removing any white space.
        TranscriptionSegment.ParseIntoArrayWS(SegmentWordArray);
        if (SegmentWordArray.Num() == 0)
        {
            UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Phrase Tree || Transcription Segment has no Word Content ||"))
            continue;
        }

        Algo::Reverse(SegmentWordArray);

        // Loop until the Segment is Empty
        while (!SegmentWordArray.IsEmpty())
        {

            FParseRecord ParseRecord = FParseRecord(ContextManager.GetContextStack());
            FParseResult ParseResult = ParsePhrase(SegmentWordArray, ParseRecord);

            ContextManager.UpdateContextStack(ParseRecord.ContextObjectStack);

            UE_LOGFMT(LogOpenAccessibilityCom, Log, "|| Phrase Tree || Segment: {0} | Result: {1} ||", SegmentCount, ParseResult.Result);

            switch (ParseResult.Result)
            {
                case PHRASE_PARSED:
                case PHRASE_PARSED_AND_EXECUTED:
                {
                    OA_LOG(LogOpenAccessibilityCom, Log, TEXT("PhraseTree Propagation"), TEXT("{Success} Phrase Tree Parsed Correctly (%s)"),
                        *ParseRecord.GetPhraseString())

                    LastVistedNode.Reset();
                    LastVistedParseRecord = FParseRecord();

                    break;
                }

                case PHRASE_REQUIRES_MORE:
                {
                    OA_LOG(LogOpenAccessibilityCom, Log, TEXT("PhraseTree Propagation"), TEXT("{Failed} Phrase Tree Propagation Requires More Segments. (%s)"),
                        *ParseRecord.GetPhraseString());

                    // Store Reach Nodes, and the ParseRecord for future propagation attempts.
                    LastVistedNode = ParseResult.ReachedNode;
                    LastVistedParseRecord = ParseRecord;
                }

                case PHRASE_REQUIRES_MORE_CORRECT_PHRASES:
                {
                    OA_LOG(LogOpenAccessibilityCom, Log, TEXT("PhraseTree Propagation"),
 TEXT("{Failed} Phrase Tree Propagation Requires More Correct Segments. (%s)"),
                        *ParseRecord.GetPhraseString())

                    LastVistedNode = ParseResult.ReachedNode;
                    LastVistedParseRecord = ParseRecord;

                    // Dirty Way of Ensuring all Segments in Transcription are Attempted.
                    if (!SegmentWordArray.IsEmpty())
                        SegmentWordArray.Pop();

                    break;
                }

                default:
                case PHRASE_UNABLE_TO_PARSE:
                {
                    OA_LOG(LogOpenAccessibilityCom, Log, TEXT("PhraseTree Propagation"),
 TEXT("{Failed} Phrase Tree Propagation Failed. (%s)"),
                        *ParseRecord.GetPhraseString())

                    // Dirty Way of Ensuring all Segments in Transcription are Attempted.
                    if (!SegmentWordArray.IsEmpty())
                        SegmentWordArray.Pop();

                    break;
                }
            }
        }

        SegmentCount++;
        SegmentWordArray.Reset();
    }
}

FParseResult FPhraseTree::ParsePhrase(TArray<FString>& InPhraseWordArray, FParseRecord& InParseRecord)
{
    if (InPhraseWordArray.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Phrase Tree || Provided Transcription
 Segment is Empty ||"));

        return FParseResult(PHRASE_NOT_PARSED);
    }

    // First give the last visited node a chance to parse the phrase.
    // due to the possibility of connecting phrases over different transcription segments.
    if (LastVistedNode != nullptr && LastVistedNode.IsValid())
    {
        TArray<FString> PhraseWordArrayCopy = TArray(InPhraseWordArray);

        FParseResult ParseResult = LastVistedNode->ParseChildren(PhraseWordArrayCopy,
 LastVistedParseRecord);
        if (ParseResult.Result == PHRASE_PARSED)
        {
            LastVistedNode.Reset();
            InParseRecord = LastVistedParseRecord;
            LastVistedParseRecord = FParseRecord();

            return ParseResult;
        }
        else if (ParseResult.Result != PHRASE_UNABLE_TO_PARSE)
        {
            return ParseResult;
        }
    }

    // Check if the Context Stack has Objects, if so propagation from the Context Root.
    if (ContextManager.HasContextObjects())
    {
        // Propagate from the Context Root, that is the Top of the Context Stack.
        return
 ContextManager.PeekContextObject()->GetContextRoot()->ParsePhraseAsContext(InPhraseWordArray,
 InParseRecord);
    }

    // Otherwise, start a new propagation entirely from the Tree Root.
    return ParseChildren(InPhraseWordArray, InParseRecord);
}

void FPhraseTree::BindBranch(const TPhraseNode& InNode)
{
    TArray<FPhraseTreeBranchBind> ToBindArray = TArray<FPhraseTreeBranchBind>();

    ToBindArray.Add(FPhraseTreeBranchBind(AsShared(), InNode));

    while (!ToBindArray.IsEmpty())
    {
        FPhraseTreeBranchBind BranchToBind = ToBindArray.Pop();

        // Check all ChildNodes to see if they are similar in purpose.
        for (auto& ChildNode : BranchToBind.StartNode->ChildNodes)
        {
            // If a ChildNode meets the same requirements as the BranchRoot,
            // then Split Bind Process to the ChildNodes.
            if (ChildNode->RequiresPhrase(BranchToBind.BranchRoot->BoundPhrase))
            {
                for (auto& BranchChildNode : BranchToBind.BranchRoot->ChildNodes)
                {
                    ToBindArray.Add(FPhraseTreeBranchBind(ChildNode, BranchChildNode));
                }

                continue;
            }
        }

        // If the Start Node has no similar children, then bind the branch to the start node.
        // Can force bind, as previous checks show no child is similar.
        BranchToBind.StartNode->BindChildNodeForce(BranchToBind.BranchRoot);
    }
}

void FPhraseTree::BindBranches(const TPhraseNodeArray& InNodes)
{
    for (const TSharedPtr<FPhraseNode>& Node : InNodes)
    {
        BindBranch(Node);
    }
}
```

5.59 ContextMenuObject.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "PhraseTree/Containers/ContextMenuObject.h"

#include "OpenAccessibilityComLogging.h"

UPhraseTreeContextMenuObject::UPhraseTreeContextMenuObject()
    : UPhraseTreeContextObject()
{

}

UPhraseTreeContextMenuObject::UPhraseTreeContextMenuObject(TSharedRef<IMenu> Menu)
    : UPhraseTreeContextObject()
{

}

UPhraseTreeContextMenuObject::~UPhraseTreeContextMenuObject()
{
    // Unbind Tick Delegate
    RemoveTickDelegate();

    if (Menu.IsValid())
        RemoveMenuDismissed(Menu.Pin().ToSharedRef());

    UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Context Menu || Destroyed ||"))
}

void UPhraseTreeContextMenuObject::Init(TSharedRef<IMenu> InMenu)
{
    this->Menu = InMenu;
    this->Window = FSlateApplication::Get().FindWidgetWindow(
        InMenu->GetContent().ToSharedRef()
    );

    BindMenuDismissed(InMenu);
    BindTickDelegate();
}

void UPhraseTreeContextMenuObject::Init(TSharedRef<IMenu> InMenu, TSharedRef<FPhraseNode>
 InContextRoot)
{
    this->Menu = InMenu;
    this->Window = FSlateApplication::Get().FindWidgetWindow(
        InMenu->GetContent().ToSharedRef()
    );

    this->ContextRoot = InContextRoot;

    BindMenuDismissed(InMenu);
    BindTickDelegate();
}

void UPhraseTreeContextMenuObject::BindTickDelegate()
{
    TickDelegate = FTickerDelegate::CreateUObject(this, &UPhraseTreeContextMenuObject::Tick);
    TickDelegateHandle = FTSTicker::GetCoreTicker().AddTicker(TickDelegate);
}

void UPhraseTreeContextMenuObject::RemoveTickDelegate()
{
    if (TickDelegateHandle != NULL)
        FTSTicker::GetCoreTicker().RemoveTicker(TickDelegateHandle);
}

void UPhraseTreeContextMenuObject::BindMenuDismissed(TSharedRef<IMenu> InMenu)
{
    MenuDismissedHandle = InMenu->GetOnMenuDismissed()
        .AddUObject(this, &UPhraseTreeContextMenuObject::OnMenuDismissed);
}

void UPhraseTreeContextMenuObject::RemoveMenuDismissed(TSharedRef<IMenu> InMenu)
{
    Menu.Pin()->GetOnMenuDismissed().Remove(MenuDismissedHandle);
}

void UPhraseTreeContextMenuObject::OnMenuDismissed(TSharedRef<IMenu> InMenu)
{
    RemoveTickDelegate();

    RemoveFromRoot();
    MarkAsGarbage();

    bIsActive = false;

    UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Context Menu || Dismissed ||"))
}
```

5.60 PhraseEnumInputNode.cpp

```
// Copyright F-Dudley. All Rights Reserved.


#include "PhraseTree/PhraseEnumInputNode.h"

#include "PhraseTree/Containers/Input/UParseEnumInput.h"

template<typename TEnum>
FPhraseEnumInputNode<TEnum>::FPhraseEnumInputNode(const TCHAR* NodeName)
    : FPhraseInputNode(NodeName)
{
    static_assert(TIsEnum<TEnum>::Value, "Passed EnumType Must be an Enum.");
};

template<typename TEnum>
FPhraseEnumInputNode<TEnum>::FPhraseEnumInputNode(const TCHAR* NodeName, TPhraseNodeArray
 InChildNodes)
    : FPhraseInputNode(NodeName, InChildNodes)
{
    static_assert(TIsEnum<TEnum>::Value, "Passed EnumType Must be an Enum");
}

template<typename TEnum>
FPhraseEnumInputNode<TEnum>::FPhraseEnumInputNode(const TCHAR* InInputString,
 TDelegate<void(FParseRecord& Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes)
    : FPhraseInputNode(InInputString, InOnPhraseParsed, InChildNodes)
{
    static_assert(TIsEnum<TEnum>::Value, "Passed EnumType Must be an Enum");
}

template<typename TEnum>
FPhraseEnumInputNode<TEnum>::FPhraseEnumInputNode(const TCHAR* InInputString, TPhraseNodeArray
 InChildNodes, TDelegate<void(int32 Input)> InOnInputRecieved)
    : FPhraseInputNode(InInputString, InChildNodes, InOnInputRecieved)
{
    static_assert(TIsEnum<TEnum>::Value, "Passed EnumType Must be an Enum");
}

template<typename TEnum>
FPhraseEnumInputNode<TEnum>::FPhraseEnumInputNode(const TCHAR* InInputString,
  TDelegate<void(FParseRecord& Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes,
  TDelegate<void(int32 Input)> InOnInputRecieved)
    : FPhraseInputNode(InInputString, InOnPhraseParsed, InChildNodes, InOnInputRecieved)
{
    static_assert(TIsEnum<TEnum>::Value, "Passed EnumType Must be an Enum");
}

template<typename TEnum>
FPhraseEnumInputNode<TEnum>::~FPhraseEnumInputNode()
{

}

template<typename TEnum>
bool FPhraseEnumInputNode<TEnum>::MeetsInputRequirements(const FString& InPhrase)
{
    UEnum* EnumPtr = StaticEnum<TEnum>();
    if (!EnumPtr)
    {
        UE_LOG(LogTemp, Error, TEXT("FPhraseEnumInputNode::MeetsInputRequirements: EnumPtr is NULL"));
        return false;
    }

    return EnumPtr->IsValidEnumName(*EnumPtr->GenerateFullEnumName(*InPhrase.ToUpper()));
}

template<typename TEnum>
bool FPhraseEnumInputNode<TEnum>::RecordInput(const FString& InInput, FParseRecord& OutParseRecord)
{
    UEnum* EnumPtr = StaticEnum<TEnum>();
    if (!EnumPtr)
    {
        UE_LOG(LogTemp, Error, TEXT("FPhraseEnumInputNode::RecordInput: EnumPtr is NULL"));
        return false;
    }

    int32 Val = EnumPtr->GetValueByNameString(EnumPtr->GenerateFullEnumName(*InInput.ToUpper()));
    if (Val == INDEX_NONE)
    {
        return false;
    }

    UParseEnumInput* ParseInput = MakeParseInput<UParseEnumInput>();
    ParseInput->SetValue(Val);
    ParseInput->SetEnumType(EnumPtr);

    OutParseRecord.AddPhraseInput(BoundPhrase, ParseInput);

    return true;
}
```

5.61 PhraseEventNode.cpp

```
// Copyright F-Dudley. All Rights Reserved.


#include "PhraseTree/PhraseEventNode.h"
#include "OpenAccessibilityComLogging.h"

FPhraseEventNode::FPhraseEventNode()
    : FPhraseNode(TEXT("EVENT_NODE"))
{
    OnPhraseParsed = TDelegate<void(FParseRecord&)>();
}

FPhraseEventNode::FPhraseEventNode(TDelegate<void(FParseRecord&)> InEvent)
    : FPhraseNode(TEXT("EVENT_NODE"), InEvent)
{

}

FPhraseEventNode::FPhraseEventNode(TFunction<void(FParseRecord&)> InEventFunction)
    : FPhraseNode(TEXT("EVENT_NODE"), TDelegate<void(FParseRecord&)>::CreateLambda(InEventFunction))
{

}

FPhraseEventNode::~FPhraseEventNode()
{

}

bool FPhraseEventNode::RequiresPhrase(const FString InPhrase)
{
    return true;
}

bool FPhraseEventNode::RequiresPhrase(const FString InPhrase, int32& OutDistance)
{
    OutDistance = 0;
    return true;
}

FParseResult FPhraseEventNode::ParsePhrase(TArray<FString>& InPhraseArray, FParseRecord&
 InParseRecord)
{
    if (OnPhraseParsed.ExecuteIfBound(InParseRecord))
    {
        return FParseResult(PHRASE_PARSED_AND_EXECUTED);
    }

    UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Unable to Execute Event ||"))

    return FParseResult(PHRASE_UNABLE_TO_PARSE, AsShared());
}
```

5.62 PhraseInputNode.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "PhraseTree/PhraseInputNode.h"
#include "PhraseTree/Utils.h"
#include "OpenAccessibilityComLogging.h"

#include "PhraseTree/Containers/Input/UParseIntInput.h"

template<typename InputType>
FPhraseInputNode<InputType>::FPhraseInputNode(const TCHAR* InInputString)
    : FPhraseNode(InInputString)
{

}

template<typename InputType>
FPhraseInputNode<InputType>::FPhraseInputNode(const TCHAR* InInputString, TPhraseNodeArray
 InChildNodes)
    : FPhraseNode(InInputString, InChildNodes)
{

}

template<typename InputType>
FPhraseInputNode<InputType>::FPhraseInputNode(const TCHAR* InInputString, TDelegate<void(FParseRecord&
 Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes)
    : FPhraseNode(InInputString, InOnPhraseParsed, InChildNodes)
{

}

template<typename InputType>
FPhraseInputNode<InputType>::FPhraseInputNode(const TCHAR* InInputString, TPhraseNodeArray
 InChildNodes, TDelegate<void(InputType Input)> InOnInputRecieved)
    : FPhraseNode(InInputString, InChildNodes)
{
    OnInputReceived = InOnInputRecieved;
}

template<typename InputType>
FPhraseInputNode<InputType>::FPhraseInputNode(const TCHAR* InInputString, TDelegate<void(FParseRecord&
 Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes, TDelegate<void(InputType Input)>
 InOnInputRecieved)
    : FPhraseNode(InInputString, InOnPhraseParsed, InChildNodes)
{
    OnInputReceived = InOnInputRecieved;
}

template<typename InputType>
FPhraseInputNode<InputType>::~FPhraseInputNode()
{

}

```

```
template<typename InputType>
bool FPhraseInputNode<InputType>::RequiresPhrase(const FString InPhrase)
{
    return MeetsInputRequirements(InPhrase);
}

template <typename InputType>
bool FPhraseInputNode<InputType>::RequiresPhrase(const FString InPhrase, int32& OutDistance)
{
    bool bMeetsRequirements = MeetsInputRequirements(InPhrase);
    OutDistance = bMeetsRequirements ? 0 : INT32_MAX;

    return bMeetsRequirements;
}

template<typename InputType>
FParseResult FPhraseInputNode<InputType>::ParsePhrase(TArray<FString>& InPhraseArray, FParseRecord&
 InParseRecord)
{
    if (InPhraseArray.Num() == 0)
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Emptied Phrase Array ||"))

        return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
    }

    if (MeetsInputRequirements(InPhraseArray.Last()))
    {
        // Get the Input String.
        FString InputToRecord = InPhraseArray.Pop();

        // Append the Input String to the Record.
        InParseRecord.AddPhraseString(InputToRecord);

        if (!InputToRecord.IsNumeric() && NumericParser::IsValidNumeric(InputToRecord, false))
        {
            NumericParser::StringToNumeric(InputToRecord, false);
        }

        if (!RecordInput(InputToRecord, InParseRecord))
        {
            UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Unable to Record Input ||"))

            return FParseResult(PHRASE_UNABLE_TO_PARSE, AsShared());
        }

        OnPhraseParsed.ExecuteIfBound(InParseRecord);

        return ParseChildren(InPhraseArray, InParseRecord);
    }

    return FParseResult(PHRASE_UNABLE_TO_PARSE, AsShared());
}

template<typename InputType>
bool FPhraseInputNode<InputType>::MeetsInputRequirements(const FString& InPhrase)
{
    return InPhrase.IsNumeric() || NumericParser::IsValidNumeric(InPhrase, false);
}

template<typename InputType>
bool FPhraseInputNode<InputType>::RecordInput(const FString& InInput, FParseRecord& OutParseRecord)
{
    return false;
}

bool FPhraseInputNode<int32>::RecordInput(const FString& InInput, FParseRecord& OutParseRecord)
{
    int32 Input = FCString::Atoi(*InInput);

    UParseIntInput* ParseInput = MakeParseInput<UParseIntInput>();
    ParseInput->SetValue(Input);

    OutParseRecord.AddPhraseInput(BoundPhrase, ParseInput);

    OnInputReceived.ExecuteIfBound(Input);

    return true;
}
```

5.63 PhraseNode.cpp

```
// Copyright F-Dudley. All Rights Reserved.
```

```
#include "PhraseTree/PhraseNode.h"
#include "PhraseTree.h"
#include "OpenAccessibilityComLogging.h"

#include "Algo/LevenshteinDistance.h"

FPhraseNode::FPhraseNode(const TCHAR* InBoundPhrase)
{
    BoundPhrase = InBoundPhrase;
    BoundPhrase.ToUpperInline();

    ChildNodes = TArray<TSharedPtr<FPhraseNode»();
}

FPhraseNode::FPhraseNode(const TCHAR* InBoundPhrase, TDelegate<void(FParseRecord& Record)>
 InOnPhraseParsed)
{
    BoundPhrase = InBoundPhrase;
    BoundPhrase.ToUpperInline();

    OnPhraseParsed = InOnPhraseParsed;
    ChildNodes = TArray<TSharedPtr<FPhraseNode»();
}

FPhraseNode::FPhraseNode(const TCHAR* InBoundPhrase, TPhraseNodeArray InChildNodes)
{
    BoundPhrase = InBoundPhrase;
    BoundPhrase.ToUpperInline();

    ChildNodes = InChildNodes;
}

FPhraseNode::FPhraseNode(const TCHAR* InBoundPhrase, TDelegate<void(FParseRecord& Record)>
 InOnPhraseParsed, TPhraseNodeArray InChildNodes)
{
    BoundPhrase = InBoundPhrase;
    BoundPhrase.ToUpperInline();

    OnPhraseParsed = InOnPhraseParsed;
    ChildNodes = InChildNodes;
}

FPhraseNode::~FPhraseNode()
{

}

bool FPhraseNode::HasLeafChild() const
{
    return bHasLeafChild;
}

bool FPhraseNode::RequiresPhrase(FString InPhrase)
{
    return InPhrase.Equals(BoundPhrase, ESearchCase::IgnoreCase) ||
 Algo::LevenshteinDistance(BoundPhrase, InPhrase) < 3;
}

bool FPhraseNode::RequiresPhrase(const FString InPhrase, int32& OutDistance)
{
    OutDistance = Algo::LevenshteinDistance(BoundPhrase, InPhrase);

    return InPhrase.Equals(BoundPhrase, ESearchCase::IgnoreCase) || OutDistance < 3;
}

FParseResult FPhraseNode::ParsePhrase(TArray<FString>& InPhraseArray,
                                      FParseRecord& InParseRecord) {
    if (InPhraseArray.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Emptied Phrase Array ||"))

        return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
    }

    // Pop the Phrase Linked to this Node.
    // Apply to the Record.
    FString LinkedPhrase = InPhraseArray.Pop();

    // Append Removed Phrase To Record.
    InParseRecord.AddPhraseString(LinkedPhrase);

    OnPhraseParsed.ExecuteIfBound(InParseRecord);

    // Pass
    return ParseChildren(InPhraseArray, InParseRecord);
}
```

```

FParseResult FPhraseNode::ParsePhraseAsContext(TArray<FString>& InPhraseWordArray, FParseRecord&
 InParseRecord)
{
    if (InPhraseWordArray.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Emptied Phrase Array ||"))

            return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
    }

    OnPhraseParsed.ExecuteIfBound(InParseRecord);

    return ParseChildren(InPhraseWordArray, InParseRecord);
}

FParseResult FPhraseNode::ParsePhraseIfRequired(TArray<FString>& InPhraseWordArray, FParseRecord&
 InParseRecord)
{
    if (RequiresPhrase(InPhraseWordArray.Last()))
    {
        return ParsePhrase(InPhraseWordArray, InParseRecord);
    }

    return FParseResult(PHRASE_UNABLE_TO_PARSE);
}

bool FPhraseNode::CanBindChild(TPhraseNode& InNode)
{
    for (auto& ChildNode : ChildNodes)
    {
        if (ChildNode->RequiresPhrase(InNode->BoundPhrase) || ChildNode->IsLeafNode())
        {
            return false;
        }
    }

    return true;
}

bool FPhraseNode::BindChildNode(TPhraseNode InNode)
{
    if (!InNode.IsValid())
        return false;

    for (auto& ChildNode : ChildNodes)
    {
        if (ChildNode->RequiresPhrase(InNode->BoundPhrase))
        {
            return ChildNode->BindChildrenNodes(InNode->ChildNodes);
        }
        else
        {
            ChildNodes.AddUnique(ChildNode);
            return true;
        }
    }

    return false;
}

bool FPhraseNode::BindChildNodeForce(TPhraseNode InNode)
{
    ChildNodes.AddUnique(InNode);

    return true;
}

bool FPhraseNode::BindChildrenNodes(TPhraseNodeArray InNodes)
{
    for (auto& InNode : InNodes)
    {
        for (auto& ChildNode : ChildNodes)
        {
            if (ChildNode->RequiresPhrase(InNode->BoundPhrase))
            {
                return ChildNode->BindChildrenNodes(InNode->ChildNodes);
            }
            else
            {
                ChildNodes.AddUnique(ChildNode);
                return true;
            }
        }
    }

    return false;
}

bool FPhraseNode::BindChildrenNodesForce(TPhraseNodeArray InNodes)
{
    for (auto& InNode : InNodes)
    {
        ChildNodes.AddUnique(InNode);
    }

    return true;
}

bool FPhraseNode::HasLeafChild()
{
    return ChildNodes.Num() == 1 && ChildNodes[0]->IsLeafNode();
}

FParseResult FPhraseNode::ParseChildren(TArray<FString>& InPhraseArray, FParseRecord& InParseRecord)
{
    if (HasLeafChild())
        return ChildNodes[0]->ParsePhrase(InPhraseArray, InParseRecord);
    if (InPhraseArray.IsEmpty())
        return FParseResult(PHRASE_REQUIRES_MORE, AsShared());

    // Below Can Be Optimized.
    // Maybe bypass the loop if Distance == 0 and Sort ChildNodes with Derrived PhraseNodes Last?

    int FoundChildIndex = -1;
    {
        int32 FoundChildDistance = INT32_MAX, CurrentDistance = INT32_MAX;

        for (int i = 0; i < ChildNodes.Num(); i++)
        {
            // Child Nodes Require Unique Phrases to Siblings.
            if (ChildNodes[i]->RequiresPhrase(InPhraseArray.Last(), CurrentDistance))
            {
                if (FoundChildDistance > CurrentDistance)
                {
                    FoundChildIndex = i;
                    FoundChildDistance = CurrentDistance;
                }
            }
        }
    }

    if (FoundChildIndex != -1)
    {
        return ChildNodes[FoundChildIndex]->ParsePhrase(InPhraseArray, InParseRecord);
    }

    /*else if (!InPhraseArray.IsEmpty())
    {
        return FParseResult(PHRASE_REQUIRES_MORE_CORRECT_PHRASES, AsShared());
    }*/

    return FParseResult(PHRASE_UNABLE_TO_PARSE, AsShared());
}
```

5.64 PhraseStringInputNode.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "PhraseTree/PhraseStringInputNode.h"

#include "PhraseTree/Containers/Input/UParseStringInput.h"

FPhraseStringInputNode::FPhraseStringInputNode(const TCHAR* InInputString)
    : FPhraseInputNode(InInputString)
{

};

FPhraseStringInputNode::FPhraseStringInputNode(const TCHAR* InInputString, TPhraseNodeArray
 InChildNodes)
    : FPhraseInputNode(InInputString, InChildNodes)
{

}

FPhraseStringInputNode::FPhraseStringInputNode(const TCHAR* InInputString,
 TDelegate<void(FParseRecord& Record)> InOnPhraseParse, TPhraseNodeArray InChildNodes)
    : FPhraseInputNode(InInputString, InOnPhraseParse, InChildNodes)
{
```

```cpp

}

FPhraseStringInputNode::FPhraseStringInputNode(const TCHAR* InInputString, TPhraseNodeArray InChildNodes, TDelegate<void(FString Input)> InOnInputRecieved)
    : FPhraseInputNode(InInputString, InChildNodes, InOnInputRecieved)
{

}

FPhraseStringInputNode::~FPhraseStringInputNode()
{

}

bool FPhraseStringInputNode::MeetsInputRequirements(const FString& InPhrase)
{
    if (InPhrase.IsEmpty())
        return false;
    else return true;
}

bool FPhraseStringInputNode::RecordInput(const FString& InInput, FParseRecord& OutParseRecord)
{
    if (InInput.IsEmpty())
        return false;

    UParseStringInput* ParseInput = MakeParseInput<UParseStringInput>();
    ParseInput->SetValue(InInput);

    OutParseRecord.AddPhraseInput(BoundPhrase, ParseInput);

    OnInputReceived.ExecuteIfBound(InInput);

    return true;
}
```

5.65 Utils.cpp

```cpp
// Copyright F-Dudley. All Rights Reserved.

#include "PhraseTree/Utils.h"
#include "OpenAccessibilityComLogging.h"


bool NumericParser::IsValidNumeric(const FString& StringToCheck, bool ConvertToUpper)
{
    return StringMappings.Contains(ConvertToUpper ? StringToCheck.ToUpper() : StringToCheck);
}

void NumericParser::StringToNumeric(FString& NumericString, bool ConvertToUpper)
{
    if (const FString* FoundMapping = StringMappings.Find(NumericString))
    {
        NumericString = ConvertToUpper ? *FoundMapping->ToUpper() : *FoundMapping;
    }
    else UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Numeric Parser || No Mapping Found for String: %s ||"), *NumericString);
}

const TMap<const FString, const FString> NumericParser::StringMappings = TMap<const FString, const FString>
{
    { TEXT("ZERO"), TEXT("0") },
    { TEXT("ONE"), TEXT("1") },
    { TEXT("TWO"), TEXT("2") },
    { TEXT("TOO"), TEXT("2") },
    { TEXT("TO"), TEXT("2") },
    { TEXT("THREE"), TEXT("3") },
    { TEXT("FOUR"), TEXT("4") },
    { TEXT("FOR"), TEXT("4") },
    { TEXT("FIVE"), TEXT("5") },
    { TEXT("SIX"), TEXT("6") },
    { TEXT("SEVEN"), TEXT("7") },
    { TEXT("EIGHT"), TEXT("8") },
    { TEXT("NINE"), TEXT("9") },
    { TEXT("TEN"), TEXT("10") },
    { TEXT("TIN"), TEXT("10") },
    { TEXT("ELEVEN"), TEXT("11") },
    { TEXT("TWELVE"), TEXT("12") },
    { TEXT("THIRTEEN"), TEXT("13") },
    { TEXT("FOURTEEN"), TEXT("14") },
    { TEXT("FIFTEEN"), TEXT("15") },
    { TEXT("SIXTEEN"), TEXT("16") },
    { TEXT("SEVENTEEN"), TEXT("17") },
    { TEXT("EIGHTEEN"), TEXT("18") },
    { TEXT("NINETEEN"), TEXT("19") },
    { TEXT("TWENTY"), TEXT("20") },
    { TEXT("THIRTY"), TEXT("30") },
    { TEXT("FORTY"), TEXT("40") },
    { TEXT("FIFTY"), TEXT("50") },
    { TEXT("SIXTY"), TEXT("60") },
    { TEXT("SEVENTY"), TEXT("70") },
    { TEXT("EIGHTY"), TEXT("80") },
    { TEXT("NINETY"), TEXT("90") },
    { TEXT("HUNDRED"), TEXT("100") },
};
```

5.66 PhraseTreeUtils.cpp

```
#include "PhraseTreeUtils.h"

#include "OpenAccessibilityComLogging.h"

UPhraseTreeUtils::UPhraseTreeUtils()
{

}

UPhraseTreeUtils::~UPhraseTreeUtils()
{

}

void UPhraseTreeUtils::RegisterFunctionLibrary(UPhraseTreeFunctionLibrary* LibraryToRegister)
{
    TSharedPtr<FPhraseTree> PhraseTreeSP = PhraseTree.Pin();
    if (!PhraseTreeSP.IsValid())
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("Cannot Register Phrase Tree Function Library
 Due To InValid Phrase Tree Reference."));
        return;
    }

    // For some reason this needs to be told directly to be kept alive,
    // even though it is a UPROPERTY TArray and should be kept alive by the UObject system.
    LibraryToRegister->AddToRoot();
    LibraryToRegister->BindBranches(PhraseTreeSP.ToSharedRef());
}
```

5.67 SocketCommunicationServer.cpp

```
// Copyright F-Dudley. All Rights Reserved.

#include "SocketCommunicationServer.h"
#include "OpenAccessibilityComLogging.h"

#include "Serialization/JsonSerializer.h"

FSocketCommunicationServer::FSocketCommunicationServer(const std::string SendAddress, std::string
 RecvAddress, const int PollTimeout)
    : SendAddress(SendAddress), RecvAddress(RecvAddress), PollTimeout(PollTimeout)
{
    Context = new zmq::context_t(1);
    if (Context == nullptr)
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("Failed to create ZMQ context"));
        return;
    }

    SendSocket = new zmq::socket_t(*Context, ZMQ_PUSH);
    if (SendSocket == nullptr)
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("Failed to create ZMQ socket"));
        return;
    }

    RecvSocket = new zmq::socket_t(*Context, ZMQ_PULL);
    if (RecvSocket == nullptr)
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("Failed to create ZMQ socket"));
        return;
    }

    Poller = new zmq::poller_t<int>();
    if (Poller == nullptr)
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("Failed to create ZMQ poller"));
        return;
    }

    SendSocket->connect(SendAddress);
    RecvSocket->bind(RecvAddress);

    Poller->add(*RecvSocket, zmq::event_flags::pollin);
}

FSocketCommunicationServer::~FSocketCommunicationServer()
{
    Poller->remove(*RecvSocket);
    delete Poller; Poller = nullptr;

    SendSocket->disconnect(SendAddress);
    SendSocket->close();
    delete SendSocket; SendSocket = nullptr;

    RecvSocket->unbind(RecvAddress);
    RecvSocket->close();
    delete RecvSocket; RecvSocket = nullptr;

    Context->shutdown();
    Context->close();
    delete Context; Context = nullptr;
}

bool FSocketCommunicationServer::EventOccured()
{
    std::vector<zmq::poller_event<int>> PollEvents(1);
    if (Poller->wait_all(PollEvents, std::chrono::milliseconds(PollTimeout)) > 0)
    {
        PollEvents.clear();
        return true;
    }

    PollEvents.clear();
    return false;
}

bool FSocketCommunicationServer::SendArrayBuffer(const float* MessageData, size_t Size, ComSendFlags
  SendFlags)
{
    auto Result = SendSocket->send(zmq::const_buffer(MessageData, Size * sizeof(float)), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"),
  Result.value(), Size * sizeof(float));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
  Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::SendArrayBuffer(const float MessageData[], ComSendFlags SendFlags)
{
    auto Result = SendSocket->send(zmq::const_buffer(MessageData, sizeof MessageData), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"),
  Result.value(), int(sizeof MessageData));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
  Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::SendArrayBuffer(const TArray<float>& ArrayMessage, ComSendFlags
  SendFlag)
```

```
{
    auto Result = SendSocket->send(zmq::const_buffer(ArrayMessage.GetData(), ArrayMessage.Num() *
 sizeof(float)), SendFlag);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"),
 Result.value(), int(ArrayMessage.Num() * sizeof(float)));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
 Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::SendArrayMessage(const float* MessageData, size_t Size, ComSendFlags
 SendFlags)
{
    auto Result = SendSocket->send(zmq::message_t(MessageData, Size * sizeof(float)), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"),
 Result.value(), Size * sizeof(float));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
 Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::SendArrayMessage(const float MessageData[], ComSendFlags SendFlags)
{
    auto Result = SendSocket->send(zmq::message_t(MessageData, sizeof MessageData), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"),
 Result.value(), int(sizeof MessageData));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
 Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::SendArrayMessage(const TArray<float>& ArrayMessage, ComSendFlags
 SendFlags)
{
    auto Result = SendSocket->send(zmq::message_t(ArrayMessage.GetData(), ArrayMessage.Num() *
 sizeof(float)), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"),
 Result.value(), int(ArrayMessage.Num() * sizeof(float)));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error
 Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::SendArrayMessageWithMeta(const float* MessageData, size_t Size, const
 TSharedRef<FJsonObject>& Metadata, ComSendFlags SendFlags)
{
    FString MetaDataString;
    if (!SerializeJSON(Metadata, MetaDataString))
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("|| Com Server: Sent Array || Failed to serialize
 metadata ||"));
```

```
        return false;
    }

    std::vector<zmq::message_t> Messages;
    Messages.push_back(zmq::message_t(*MetaDataString, MetaDataString.Len() * sizeof(TCHAR)));
    Messages.push_back(zmq::message_t(MessageData, Size * sizeof(float)));

    auto Result = zmq::send_multipart(*SendSocket, Messages, SendFlags);

    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"), Result.value(), Size * sizeof(float));
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::SendArrayMessageWithMeta(const float MessageData[], const TSharedRef<FJsonObject>& Metadata, ComSendFlags SendFlags)
{
    FString MetaDataString;
    if (!SerializeJSON(Metadata, MetaDataString))
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("|| Com Server: Sent Array || Failed to serialize metadata ||"));
        return false;
    }

    std::vector<zmq::message_t> Messages;
    Messages.push_back(zmq::message_t(*MetaDataString, MetaDataString.Len() * sizeof(TCHAR)));
    Messages.push_back(zmq::message_t(MessageData, sizeof MessageData));

    auto Result = zmq::send_multipart(*SendSocket, Messages, SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d bytes"), Result.value(), int(sizeof MessageData));

        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::SendArrayMessageWithMeta(const TArray<float>& ArrayMessage, const TSharedRef<FJsonObject>& Metadata, ComSendFlags SendFlags)
{
    FString MetaDataString;
    if (!SerializeJSON(Metadata, MetaDataString))
    {
        UE_LOG(LogOpenAccessibilityCom, Error, TEXT("|| Com Server: Sent Array || Failed to serialize metadata ||"));
        return false;
    }

    std::vector<zmq::message_t> Messages;
    Messages.push_back(zmq::message_t(*MetaDataString, MetaDataString.Len() * sizeof(TCHAR)));
    Messages.push_back(zmq::message_t(ArrayMessage.GetData(), ArrayMessage.Num() * sizeof(float)));

    auto Result = zmq::send_multipart(*SendSocket, Messages, SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent Array || Sent %d of %d Messages"), Result.value(), Messages.size());

        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent Array || EAGAIN Error Occured ||"));

        return true;
```

```cpp
    }

    return false;
}

bool FSocketCommunicationServer::SendStringBuffer(const std::string StringMessage, ComSendFlags SendFlags)
{
    auto Result = SendSocket->send(zmq::const_buffer(StringMessage.c_str(), StringMessage.size()), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent String || Sent %d of %d bytes"), Result.value(), StringMessage.size());
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent String || EAGAIN Error Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::SendJsonBuffer(const std::string JsonMessage, ComSendFlags SendFlags)
{
    auto Result = SendSocket->send(zmq::const_buffer(JsonMessage.c_str(), JsonMessage.size()), SendFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Sent JSON || Sent %d of %d bytes"), Result.value(), JsonMessage.size());
        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Sent JSON || EAGAIN Error Occured ||"));
        return true;
    }

    return false;
}



template <typename T>
bool FSocketCommunicationServer::RecvArray(TArray<T>& OutArrayData, size_t Size, ComRecvFlags RecvFlags)
{
    zmq::message_t RecvMessage;

    auto Result = RecvSocket->recv(RecvMessage, RecvFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Recv Array || Recv %d bytes"), Result.value());

        OutArrayData.Append(RecvMessage.data<T>(), Result.value());

        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Recv Array || EAGAIN Error Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::RecvString(FString& OutStringMessage, ComRecvFlags RecvFlags)
{
    zmq::message_t RecvMessage;

    auto Result = RecvSocket->recv(RecvMessage, RecvFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Recv String || Recv %d bytes"), Result.value());

        OutStringMessage = FString(Result.value(), UTF8_TO_TCHAR(RecvMessage.data()));

        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {

        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Recv String || EAGAIN Error
Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::RecvJson(FString& OutJsonMessage, ComRecvFlags RecvFlags)
{
    zmq::message_t RecvMessage;

    auto Result = RecvSocket->recv(RecvMessage, RecvFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Recv JSON || Recv %d bytes"),
Result.value());

        OutJsonMessage = FString(Result.value(), UTF8_TO_TCHAR(RecvMessage.data()));

        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Recv JSON || EAGAIN Error
Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::RecvStringMultipart(TArray<FString>& OutMessages, ComRecvFlags
RecvFlags)
{
    std::vector<zmq::message_t> RecvMessages;

    auto Result = zmq::recv_multipart(*RecvSocket, std::back_inserter(RecvMessages), RecvFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Recv Multipart || Recv %d
messages"), Result.value());

        for (auto& Message : RecvMessages)
        {
            OutMessages.Add(FString(Message.size(), UTF8_TO_TCHAR(Message.data())));
        }

        return true;
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Recv Multipart || EAGAIN Error
Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::RecvStringMultipartWithMeta(TArray<FString>& OutMessages,
TSharedPtr<FJsonObject>& OutMetadata, ComRecvFlags RecvFlag)
{
    std::vector<zmq::message_t> RecvMessages;
    if (!RecvMultipartWithMeta(RecvMessages, OutMetadata, RecvFlag))
        return false;

    for (auto& Message : RecvMessages)
    {
        OutMessages.Add(FString(Message.size(), UTF8_TO_TCHAR(Message.data())));
    }

    return true;
}

bool FSocketCommunicationServer::RecvMultipartWithMeta(std::vector<zmq::message_t>&
OutMultipartMessages, TSharedPtr<FJsonObject>& OutMetadata, ComRecvFlags RecvFlags)
{
    auto Result = zmq::recv_multipart(*RecvSocket, std::back_inserter(OutMultipartMessages),
RecvFlags);
    if (Result.has_value())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Com Server: Recv Multipart || Recv %d
messages"), Result.value());

        // Pop Metadata Messages from the Front of Array.
        zmq::message_t MetadataMessage = MoveTempIfPossible(OutMultipartMessages[0]);
        OutMultipartMessages.erase(OutMultipartMessages.begin());

        if (DeserializeJSON(FString(UTF8_TO_TCHAR(MetadataMessage.data()), MetadataMessage.size()), OutMetadata))
        {
            return true;
        }
        else
        {
            UE_LOG(LogOpenAccessibilityCom, Error, TEXT("|| Com Server: Recv Multipart || Failed to deserialize metadata ||"));
            return false;
        }
    }
    else if (zmq_errno() == EAGAIN)
    {
        UE_LOG(LogOpenAccessibilityCom, Warning, TEXT("|| Com Server: Recv Multipart || EAGAIN Error Occured ||"));
        return true;
    }

    return false;
}

bool FSocketCommunicationServer::SerializeJSON(const TSharedRef<FJsonObject>& InJsonObject, FString& OutJsonString)
{
    return FJsonSerializer::Serialize(InJsonObject, TJsonWriterFactory<TCHAR>::Create(&OutJsonString));
}

bool FSocketCommunicationServer::DeserializeJSON(const FString& InJsonString, TSharedPtr<FJsonObject>& OutJsonObject)
{
    return FJsonSerializer::Deserialize(TJsonReaderFactory<TCHAR>::Create(InJsonString), OutJsonObject);
}
```

5.68 UBAudioCapture.cpp

```
// Copyright F-Dudley. All Rights Reserved.


#include "UBAudioCapture.h"

UBAudioCapture::UBAudioCapture() : UAudioCapture()
{

}

UBAudioCapture::~UBAudioCapture()
{
}

bool UBAudioCapture::OpenDefaultAudioStream(int32 OverrideSampleRate, int32 OverrideInputChannels)
{
    if (!AudioCapture.IsStreamOpen())
    {
        if (!AudioCapture.IsStreamOpen())
        {
            Audio::FOnAudioCaptureFunction OnCapture = [this](const void* AudioData, int32 NumFrames, int32 InNumChannels, int32 InSampleRate, double StreamTime, bool bOverFlow)
                {
                    OnGeneratedAudio((const float*)AudioData, NumFrames * InNumChannels);
                };

            // Start the stream here to avoid hitching the audio render thread.
            Audio::FAudioCaptureDeviceParams Params;
            if (OverrideSampleRate != NULL)
                Params.SampleRate = OverrideSampleRate;
            if (OverrideInputChannels != NULL)
                Params.NumInputChannels = OverrideInputChannels;


            if (AudioCapture.OpenAudioCaptureStream(Params, MoveTemp(OnCapture), 1024))
            {
                // If we opened the capture stream succesfully, get the capture device info and initialize the UAudioGenerator
                Audio::FCaptureDeviceInfo Info;
                if (AudioCapture.GetCaptureDeviceInfo(Info))
                {
                    Init(
                        OverrideSampleRate != NULL ? OverrideSampleRate : Info.PreferredSampleRate ,
                        OverrideInputChannels != NULL ? OverrideInputChannels : Info.InputChannels
                    );

                    return true;
                }
            }
        }

        return false;
    }

    return false;
}
```

5.69 AudioManager.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "AudioCapture.h"
#include "Sound/SampleBufferIO.h"
#include "Delegates/DelegateCombinations.h"
#include "AudioDeviceNotificationSubsystem.h"

#include "AudioManager.generated.h"

USTRUCT()
struct FAudioManagerSettings
{
    GENERATED_BODY()

public:
    FAudioManagerSettings()
    {
        // Default Settings
        LevelThreshold = -2.5f;
        SaveName = FString("Captured_User_Audio");
        SavePath = FString("./OpenAccessibility/Audioclips/");
    }

    // The Threshold for incoming audio to be considered as input.
    UPROPERTY(Config, EditAnywhere, Category = "OpenAccessibility/Audio Manager")
    float LevelThreshold;

    UPROPERTY(Config, EditAnywhere, Category = "OpenAccessibility/Audio Manager")
    FString SaveName;

    UPROPERTY(Config, EditAnywhere, Category = "OpenAccessibility/Audio Manager")
    FString SavePath;
};


UCLASS(BlueprintType, Blueprintable, Config = OpenAccessibility)
class OPENACCESSIBILITYCOMMUNICATION_API UAudioManager : public UObject
{
    GENERATED_BODY()

public:
    UAudioManager();
    virtual ~UAudioManager();

    void StartCapturingAudio();

    void StopCapturingAudio();

    void PRIVATE_OnAudioGenerate(const float* InAudio, int32 NumSamples);

    void SaveAudioBufferToWAV(const FString& FilePath);

    bool IsCapturingAudio() const { return bIsCapturingAudio; }

    int32 GetAudioCaptureSampleRate() const { return AudioCapture->GetSampleRate(); }

    int32 GetAudioCaptureNumChannels() const { return AudioCapture->GetNumChannels(); }

    void OnDefaultDeviceChanged(EAudioDeviceChangedRole ChangedRole, FString DeviceID);

private:

    void RegisterAudioGenerator();

    void UnregisterAudioGenerator();

public:

    UPROPERTY(Config, EditAnywhere, Category = "OpenAccessibility/Audio Manager")
    FAudioManagerSettings Settings;

    TDelegate<void(const TArray<float>)> OnAudioReadyForTranscription;

private:

    // Audio Capture
    bool bIsCapturingAudio = false;

    UPROPERTY(EditDefaultsOnly, Category = "OpenAccessibility/Audio Capture")
    class UAudioCapture* AudioCapture;
    TArray<float> AudioBuffer;

    FAudioGeneratorHandle OnAudioGenerateHandle;

    FDelegateHandle OnDefaultDeviceChangedHandle;

    // Audio Saving
    Audio::FSoundWavePCMWriter* FileWriter;
};
```

5.70 OpenAccessibilityComLogging.h

```
// Copyright Epic Games, Inc. All Rights Reserved.

#pragma once

DECLARE_LOG_CATEGORY_EXTERN(LogOpenAccessibilityCom, Log, All);

DEFINE_LOG_CATEGORY(LogOpenAccessibilityCom);
```

5.71 OpenAccessibilityCommunication.h

```
// Copyright Epic Games, Inc. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "Modules/ModuleManager.h"
#include "Modules/ModuleInterface.h"
#include "Delegates/DelegateCombinations.h"

#include "PhraseTree.h"
#include "PhraseTreeUtils.h"

//UDELEGATE()
//DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FTranscriptionRecievedSignature, const TArray<FString>,
 InTranscription);

class FOpenAccessibilityCommunicationModule : public IModuleInterface
{

public:

    virtual void StartupModule() override;
    virtual void ShutdownModule() override;

    virtual bool SupportsDynamicReloading() override
    {
        return false;
    }
    static FOpenAccessibilityCommunicationModule& Get()
    {
        return
 FModuleManager::GetModuleChecked<FOpenAccessibilityCommunicationModule>("OpenAccessibilityCommunication");
    }

    bool Tick(const float DeltaTime);

    void HandleKeyDownEvent(const FKeyEvent& InKeyEvent);

    void TranscribeWaveForm(TArray<float> AudioBufferToTranscribe);

private:

    void BuildPhraseTree();

    void RegisterConsoleCommands();

    void UnregisterConsoleCommands();

    void LoadZMQDLL();

    void UnloadZMQDLL();
public:

    TMulticastDelegate<void(TArray<FString>)> OnTranscriptionRecieved;

    class UAudioManager* AudioManager;

    TSharedPtr<class FSocketCommunicationServer> SocketServer;

    TSharedPtr<FPhraseTree> PhraseTree;

    class UPhraseTreeUtils* PhraseTreeUtils;

private:

    TArray<float> PrevAudioBuffer;

    FTickerDelegate TickDelegate;
    FTSTicker::FDelegateHandle TickDelegateHandle;

    FDelegateHandle PhraseTreePhraseRecievedHandle;

    FDelegateHandle KeyDownEventHandle;

    void* ZMQDllHandle;

    TArray<IConsoleCommand*> ConsoleCommands;
};
```

5.72 PhraseTree.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "PhraseTree/PhraseNode.h"
#include "PhraseTree/Containers/ParseRecord.h"
#include "PhraseTree/Containers/ContextObject.h"

enum EPhraseTreeBranchBindResult : uint8_t
{
    BRANCH_NOT_BOUND,

    BRANCH_BOUND,
    BRANCH_SPLIT
};

struct OPENACCESSIBILITYCOMMUNICATION_API FPhraseTreeBranchBind
{
    FPhraseTreeBranchBind()
    {

    }

    FPhraseTreeBranchBind(TPhraseNode InRootNode, TPhraseNode InBranchRoot)
    {
        StartNode = InRootNode;
        BranchRoot = InBranchRoot;
    }

    ~FPhraseTreeBranchBind()
    {
        StartNode.Reset();
        BranchRoot.Reset();
    }

    TPhraseNode StartNode;

    TPhraseNode BranchRoot;
};

struct OPENACCESSIBILITYCOMMUNICATION_API FPhraseTreeContextManager
{
friend class FPhraseTree;

public:

    FPhraseTreeContextManager()
    {

    }

    ~FPhraseTreeContextManager()
    {

    }

    // Context Stack Management

    void IsEmpty()
    {
        this->ContextObjectStack.IsEmpty();
    }

    bool HasContextObjects()
    {
        return this->ContextObjectStack.Num() > 0;
    }

    bool HasContextObject(UPhraseTreeContextObject* InContextObject)
    {
        return this->ContextObjectStack.Contains(InContextObject);
    }

    TArray<UPhraseTreeContextObject*> GetContextStack()
    {
        return this->ContextObjectStack;
    }

    // Context Stack Ammendments

    void PeekContextObject(UPhraseTreeContextObject* OutContextObject)
    {
        OutContextObject = this->ContextObjectStack.Top();
    }

    UPhraseTreeContextObject* PeekContextObject()
    {
        return this->ContextObjectStack.Top();
    }

    void PushContextObject(UPhraseTreeContextObject* InContextObject)
    {
        this->ContextObjectStack.Push(InContextObject);
    }

    void PopContextObject()
    {
        this->ContextObjectStack.Pop();
    }

    template<class CastToContextType>
    void PopContextObject(CastToContextType* OutContextObject)
    {
        OutContextObject = Cast<CastToContextType>(this->ContextObjectStack.Pop());
    }

    void PopContextObject(UPhraseTreeContextObject* OutContextObject)
    {
        OutContextObject = this->ContextObjectStack.Pop();
    }

private:

    void UpdateContextStack(TArray<UPhraseTreeContextObject*> InContextObjectStack)
    {
        this->ContextObjectStack = InContextObjectStack;

        FilterContextStack();
    }

    // Context Stack Filtering

    void FilterContextStack()
    {
        bool bRemoveDerivedContextObjects = false;

        int i = this->ContextObjectStack.Num() - 1;
        if (i < 0)
            return;

        UPhraseTreeContextObject* CurrObj = nullptr;

        do
        {
            CurrObj = this->ContextObjectStack[i];

            if (CurrObj != nullptr && CurrObj->GetIsActive())
            {
                i--;
                continue;
            }

            if (CurrObj->IsValidLowLevel())
            {
                CurrObj->RemoveFromRoot();
                CurrObj->MarkAsGarbage();
            }

            this->ContextObjectStack.RemoveAt(i);
            i--;

        } while (i > 0);

        CurrObj = nullptr;
    }

private:

    TArray<UPhraseTreeContextObject*> ContextObjectStack;

};

class OPENACCESSIBILITYCOMMUNICATION_API FPhraseTree : public FPhraseNode
{
public:
    FPhraseTree();
    ~FPhraseTree();

    FPhraseTreeContextManager& GetContextManager() {
        return ContextManager;
    }

    bool Tick(float DeltaTime);

    // FPhaseNode Implementation
    virtual FParseResult ParsePhrase(TArray<FString>& InPhraseWordArray, FParseRecord& InParseRecord)
 override;
    // End FPhaseNode Implementation

    void BindBranch(const TPhraseNode& InNode);

    void BindBranches(const TPhraseNodeArray& InNodes);

    void ParseTranscription(TArray<FString> InTranscriptionSegments);

private:

    TSharedPtr<FPhraseNode> LastVistedNode;

    FParseRecord LastVistedParseRecord;

    FPhraseTreeContextManager ContextManager;

    FTSTicker::FDelegateHandle TickDelegateHandle;
};
```

5.73 ContextMenuObject.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "Framework/Application/IMenu.h"

```

```
#include "PhraseTree/Containers/ContextObject.h"

#include "ContextMenuObject.generated.h"

UCLASS()
class OPENACCESSIBILITYCOMMUNICATION_API UPhraseTreeContextMenuObject : public
 UPhraseTreeContextObject
{
    GENERATED_BODY()

public:

    UPhraseTreeContextMenuObject();
    UPhraseTreeContextMenuObject(TSharedRef<IMenu> Menu);

    virtual ~UPhraseTreeContextMenuObject();

    virtual void Init(TSharedRef<IMenu> InMenu);

    virtual void Init(TSharedRef<IMenu> InMenu, TSharedRef<FPhraseNode> InContextRoot);

    virtual bool Tick(float DeltaTime) { return true; };

    virtual bool Close() override
    {
        RemoveTickDelegate();
        Menu.Pin()->Dismiss();

        return true;
    };

    void BindTickDelegate();

    void RemoveTickDelegate();

    void BindMenuDismissed(TSharedRef<IMenu> InMenu);

    void RemoveMenuDismissed(TSharedRef<IMenu> InMenu);

    virtual void SetMenu(TSharedRef<IMenu> InMenu)
    {
        Menu = InMenu;
    }

    virtual void ScaleMenu(const float ScaleFactor) {};

protected:

    TSharedPtr<SWindow> GetWindow()
    {
        return Menu.Pin()->GetOwnedWindow();
    }

    void OnMenuDismissed(TSharedRef<IMenu> Menu);

public:

    TWeakPtr<IMenu> Menu;

    TWeakPtr<SWindow> Window;

private:

    // Ticker Components

    FTickerDelegate TickDelegate;
    FTSTicker::FDelegateHandle TickDelegateHandle;

    FDelegateHandle MenuDismissedHandle;
};
```

5.74 ContextObject.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "ContextObject.generated.h"

class FPhraseNode;

```

```
UCLASS(Abstract)
class OPENACCESSIBILITYCOMMUNICATION_API UPhraseTreeContextObject : public UObject
{
    GENERATED_BODY()

public:

    UPhraseTreeContextObject()
        : UObject()
    {

    }

    ~UPhraseTreeContextObject()
    {

    }

    virtual bool Close() { return true; }

    void SetContextRootNode(TSharedRef<FPhraseNode> InRootNode)
    {
        ContextRoot = InRootNode;
    }

    TSharedPtr<FPhraseNode> GetContextRoot()
    {
        return ContextRoot.Pin();
    }

    const bool GetIsActive()
    {
        return bIsActive;
    }

protected:

    bool bIsActive = true;

    TWeakPtr<FPhraseNode> ContextRoot;
};
```

5.75 InputContainers.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

UENUM()
enum class EPhrasePositionalInput : uint8
{
    TOP,
    MIDDLE,
    BOTTOM,
    LEFT,
    RIGHT,
    CENTER
};

UENUM()
enum class EPhraseDirectionalInput : int8
{
    UP,
    DOWN,
    LEFT,
    RIGHT,
    FORWARD,
    BACKWARD
};

UENUM()
enum class EPhrase2DDirectionalInput : int8
{
    UP = EPhraseDirectionalInput::UP,
    DOWN = EPhraseDirectionalInput::DOWN,
    LEFT = EPhraseDirectionalInput::LEFT,
    RIGHT = EPhraseDirectionalInput::RIGHT,
};

UENUM()
enum class EPhraseScrollInput : uint8
```

```
{
    UP, // 0
    DOWN, // 1
    TOP, // 2
    BOTTOM // 3
};
```

5.76 UParseEnumInput.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "UParseIntInput.h"

#include "UParseEnumInput.generated.h"

UCLASS()
class OPENACCESSIBILITYCOMMUNICATION_API UParseEnumInput : public UParseIntInput
{
    GENERATED_BODY()

public:

    UParseEnumInput() = default;
    virtual ~UParseEnumInput()
    {
        delete EnumType;
    };

    void SetEnumType(UEnum* InEnumType)
    {
        EnumType = InEnumType;
    }

    void GetEnumType(UEnum*& OutEnumType)
    {
        OutEnumType = EnumType;
    }

    UEnum* GetEnumType()
    {
        return EnumType;
    }

protected:
    UEnum* EnumType;

};
```

5.77 UParseInput.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "UObject/Object.h"

#include "UParseInput.generated.h"

UCLASS()
class OPENACCESSIBILITYCOMMUNICATION_API UParseInput : public UObject
{
    GENERATED_BODY()

public:

    UParseInput() = default;
    virtual ~UParseInput()
    {

    };
};

// Input Constructor Functions

template<class ParseInputType>
```

```
[[nodiscard]] FORCEINLINE ParseInputType* MakeParseInput()
{
    ParseInputType* NewObj = NewObject<ParseInputType>();
    NewObj->AddToRoot();

    return NewObj;
}
```

5.78 UParseIntInput.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "UParseInput.h"

#include "UParseIntInput.generated.h"

UCLASS()
class OPENACCESSIBILITYCOMMUNICATION_API UParseIntInput : public UParseInput
{
    GENERATED_BODY()

public:

    UParseIntInput() = default;
    virtual ~UParseIntInput()
    {

    };

    void SetValue(int32 InValue)
    {
        Value = InValue;
    }

    void GetValue(int32& OutValue)
    {
        OutValue = Value;
    }

    int32 GetValue()
    {
        return Value;
    }

protected:

    int32 Value;

};
```

5.79 UParseStringInput.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "UParseInput.h"

#include "UParseStringInput.generated.h"

UCLASS()
class OPENACCESSIBILITYCOMMUNICATION_API UParseStringInput : public UParseInput
{
    GENERATED_BODY()

public:

    UParseStringInput() = default;
    virtual ~UParseStringInput()
    {

    };

    void SetValue(FString InValue)
    {
        Value = InValue;
    }

    void GetValue(FString& OutValue)
    {
        OutValue = Value;
    }

    FString GetValue()
    {
        return Value;
    }

protected:

    FString Value;
};
```

5.80 ParseRecord.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "Input/UParseInput.h"
#include "PhraseTree/Containers/ContextObject.h"

#include "ParseRecord.generated.h"

USTRUCT(BlueprintType)
struct OPENACCESSIBILITYCOMMUNICATION_API FParseRecord
{
    GENERATED_BODY()

public:
    friend class FPhraseTree;

    FParseRecord()
    {
        PhraseInputs = TMultiMap<FString, UParseInput*>();
        ContextObjectStack = TArray<UPhraseTreeContextObject*>();
    }

    FParseRecord(TArray<UPhraseTreeContextObject*> InContextObjects)
    {
        PhraseInputs = TMultiMap<FString, UParseInput*>();
        ContextObjectStack = InContextObjects;
    }

    ~FParseRecord()
    {
        PhraseInputs.Empty();
    }

    // -- Phrase String

    FString GetPhraseString() const
    {
        return FString::Join(PhraseRecord, TEXT(" "));
    }

    void AddPhraseString(FString StringToRecord)
    {
        PhraseRecord.Add(StringToRecord);
    }

    // --


    UParseInput* GetPhraseInput(const FString& InString)
    {
        // Check If The Phrase Exits
        // This Error Will Be Thrown If: InString Is In Correct (Requires UpperCase) or The Phrase
 Does Not Exist.
        check(PhraseInputs.Contains(InString))

        return *PhraseInputs.Find(InString);
    }

    template<class CastToType>
    CastToType* GetPhraseInput(const FString& InString)
    {
        // Check If The Phrase Exits
        // This Error Will Be Thrown If: InString Is In Correct (Requires UpperCase) or The Phrase
Does Not Exist.
        check(PhraseInputs.Contains(InString))

        return Cast<CastToType>(*PhraseInputs.Find(InString));
    }

    void GetPhraseInput(const FString& InString, UParseInput* OutInput)
    {
        // Check If The Phrase Exits
        // This Error Will Be Thrown If: InString Is In Correct (Requires UpperCase) or The Phrase
Does Not Exist.
        check(PhraseInputs.Contains(InString))

        OutInput = *PhraseInputs.Find(InString);
    }

    template<class CastToType>
    void GetPhraseInput(const FString& InString, CastToType* OutInput)
    {
        // Check If The Phrase Exits
        // This Error Will Be Thrown If: InString Is In Correct (Requires UpperCase) or The Phrase
Does Not Exist.
        check(PhraseInputs.Contains(InString))

        OutInput = Cast<CastToType>(*PhraseInputs.Find(InString));
    }

    // -- GetPhraseInputs

    void GetPhraseInputs(const FString& InString, TArray<UParseInput*>& OutInputs, const bool
MaintainOrder = true)
    {
        // Check If The Phrase Exits
        // This Error Will Be Thrown If: InString Is In Correct (Requires UpperCase) or The Phrase
Does Not Exist.
        check(PhraseInputs.Contains(InString))

        PhraseInputs.MultiFind(InString, OutInputs, MaintainOrder);
    }

    TArray<UParseInput*> GetPhraseInputs(const FString& InString, const bool MaintainOrder = true)
    {
        // Check If The Phrase Exits
        // This Error Will Be Thrown If: InString Is In Correct (Requires UpperCase) or The Phrase
Does Not Exist.
        check(PhraseInputs.Contains(InString))

        TArray<UParseInput*> OutInputs;

        PhraseInputs.MultiFind(InString, OutInputs, MaintainOrder);

        return OutInputs;
    }

    // -- PhraseInput

    void AddPhraseInput(const FString& InString, UParseInput* InInput)
    {
        PhraseInputs.Add(InString.ToUpper(), InInput);
    }

    void RemovePhraseInput(const FString& InString)
    {
        PhraseInputs.Remove(InString);
    }

    // -- ContextObject Stack Modification

    void PushContextObj(UPhraseTreeContextObject* InObject)
    {
        this->ContextObjectStack.Push(InObject);
    }

    void PopContextObj()
    {
        if (ContextObjectStack.IsEmpty())
            return;

        this->ContextObjectStack.Pop();
    }

    void PopContextObj(UPhraseTreeContextObject* OutObject)
    {
        if (ContextObjectStack.IsEmpty())
        {
            OutObject = nullptr;
            return;
        }

        OutObject = this->ContextObjectStack.Pop();
    }

    void RemoveContextObj(UPhraseTreeContextObject* InObject)
    {
        this->ContextObjectStack.Remove(InObject);
    }

    // -- HasContextObj

    bool HasContextObj()
    {
        return this->ContextObjectStack.Num() > 0;
    }

    bool HasContextObj(UPhraseTreeContextObject* InObject)
    {
        return HasContextObj() && this->ContextObjectStack.Contains(InObject);
    }

    // -- GetContextObj

    UPhraseTreeContextObject* GetContextObj()
    {
        if (ContextObjectStack.IsEmpty())
            return nullptr;

        return this->ContextObjectStack.Last();
    }

    void GetContextObj(UPhraseTreeContextObject* OutObject)
    {
        if (ContextObjectStack.IsEmpty())
        {
            OutObject = nullptr;
            return;
        }

        OutObject = this->ContextObjectStack.Last();
    }

    template<class CastToType>
    CastToType* GetContextObj()
    {
        if (ContextObjectStack.IsEmpty())
            return nullptr;

        return Cast<CastToType>(this->ContextObjectStack.Last());
    }

    template<class CastToType>
    void GetContextObj(CastToType* OutObject)
    {
        if (ContextObjectStack.IsEmpty())
        {
            OutObject = nullptr;
            return;
        }

        OutObject = Cast<CastToType>(this->ContextObjectStack.Last());
    }

    void GetContextStack(TArray<UPhraseTreeContextObject*> OutContextStack)
    {
        OutContextStack = ContextObjectStack;
    }

    TArray<UPhraseTreeContextObject*> GetContextStack()
    {
        return ContextObjectStack;
    }

protected:

    TArray<UPhraseTreeContextObject*> ContextObjectStack = TArray<UPhraseTreeContextObject*>();

    TArray<FString> PhraseRecord;

    TMultiMap<FString, UParseInput*> PhraseInputs;

};
```

5.81 ParseResult.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

class FPhraseNode;

typedef TSharedPtr<FPhraseNode> TPhraseNode;

typedef TArray<TPhraseNode> TPhraseNodeArray;


enum OPENACCESSIBILITYCOMMUNICATION_API PhrasePropogationType : uint8_t
{
    PHRASE_NOT_PARSED = 0,

    PHRASE_UNABLE_TO_PARSE = 1,

    PHRASE_REQUIRES_MORE = 2,

    PHRASE_REQUIRES_MORE_CORRECT_PHRASES = 3,

    PHRASE_PARSED = 4,

    PHRASE_PARSED_AND_EXECUTED = 5,
};

struct OPENACCESSIBILITYCOMMUNICATION_API FParseResult
{
    FParseResult()
    {
        Result = PHRASE_NOT_PARSED;
    }

    FParseResult(PhrasePropogationType InResult)
    {
        Result = InResult;
    }

    FParseResult(PhrasePropogationType InResult, TSharedPtr<FPhraseNode> InReachedNode)
    {
        Result = InResult;
        ReachedNode = InReachedNode;
    }

public:

    uint8_t Result;

    TSharedPtr<FPhraseNode> ReachedNode;
};
```

5.82 IPhraseContextNode.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "PhraseTree/Containers/ContextObject.h"

class IPhraseContextNodeBase
{
protected:

    virtual bool HasContextObject(TArray<UPhraseTreeContextObject*> InContextObjects) const = 0;

    virtual UPhraseTreeContextObject* CreateContextObject(FParseRecord& Record) = 0;

    virtual void ConstructContextChildren(TArray<TSharedPtr<class FPhraseNode»& InChildNodes) = 0;
};
```

5.83 PhraseContextMenuNode.h

```
// Copyright F-Dudley. All Rights Reserved.
```

```cpp

#pragma once

#include "CoreMinimal.h"

#include "PhraseTree/PhraseNode.h"
#include "PhraseTree/IPhraseContextNode.h"
#include "PhraseTree/PhraseEventNode.h"
#include "PhraseTree/Containers/ContextMenuObject.h"
#include "OpenAccessibilityComLogging.h"

template<typename ContextMenuType = UPhraseTreeContextMenuObject>
class FPhraseContextMenuNode : public FPhraseNode, public IPhraseContextNodeBase
{
public:

    static_assert(std::is_base_of_v<UPhraseTreeContextMenuObject, ContextMenuType>, "ContextType must
 be a subclass of UPhraseTreeContextMenuObject");

    FPhraseContextMenuNode(const TCHAR* InInputString)
        : FPhraseNode(InInputString)
        , ContextMenuScalar(1.0f)
    {
        this->ChildNodes = TPhraseNodeArray();
    };

    FPhraseContextMenuNode(const TCHAR* InInputString, TPhraseNodeArray InChildNodes)
        : FPhraseNode(InInputString)
        , ContextMenuScalar(1.0f)
    {
        ConstructContextChildren(InChildNodes);
    };

    FPhraseContextMenuNode(const TCHAR* InInputString, TDelegate<TSharedPtr<IMenu>(FParseRecord&
 Record)> InOnGetMenu, TPhraseNodeArray InChildNodes)
        : FPhraseNode(InInputString)
        , ContextMenuScalar(1.0f)
        , OnGetMenu(InOnGetMenu)
    {
        ConstructContextChildren(InChildNodes);
    };

    FPhraseContextMenuNode(const TCHAR* InInputString, const float InMenuScalar, TPhraseNodeArray
 InChildNodes)
        : FPhraseNode(InInputString)
        , ContextMenuScalar(InMenuScalar)
    {
        ConstructContextChildren(InChildNodes);
    };

    FPhraseContextMenuNode(const TCHAR* InInputString, const float InMenuScalar,
 TDelegate<TSharedPtr<IMenu>(FParseRecord& Record)> InOnGetMenu, TPhraseNodeArray InChildNodes)
        : FPhraseNode(InInputString)
        , ContextMenuScalar(InMenuScalar)
        , OnGetMenu(InOnGetMenu)
    {
        ConstructContextChildren(InChildNodes);
    }

    FPhraseContextMenuNode(const TCHAR* InInputString, const float InMenuScalar,
 TDelegate<void(FParseRecord& Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes)
        : FPhraseNode(InInputString, InOnPhraseParsed)
        , ContextMenuScalar(InMenuScalar)
    {
        ConstructContextChildren(InChildNodes);
    }

    FPhraseContextMenuNode(const TCHAR* InInputString, const float InMenuScalar,
 TDelegate<TSharedPtr<IMenu>(FParseRecord& Record)> InOnGetMenu, TDelegate<void(FParseRecord& Record)>
 InOnPhraseParsed, TPhraseNodeArray InChildNodes)
        : FPhraseNode(InInputString, InOnPhraseParsed)
        , ContextMenuScalar(InMenuScalar)
        , OnGetMenu(InOnGetMenu)
    {
        ConstructContextChildren(InChildNodes);
    }

    ~FPhraseContextMenuNode()
    {

    }

    // FPhraseNode Implementation

    virtual FParseResult ParsePhrase(TArray<FString>& InPhraseWordArray, FParseRecord& InParseRecord)
 override;

```

```
    virtual FParseResult ParsePhraseAsContext(TArray<FString>& InPhraseWordArray, FParseRecord&
 InParseRecord) override;

    // End FPhraseNode Implementation

protected:

    // FPhraseContextNodeBase Implementation

    virtual bool HasContextObject(TArray<UPhraseTreeContextObject*> InContextObjects) const override;

    virtual UPhraseTreeContextObject* CreateContextObject(FParseRecord& Record) override;

    virtual void ConstructContextChildren(TPhraseNodeArray& InChildNodes) override;

    // End FPhraseContextNode Implementation

protected:

    const float ContextMenuScalar;

    TDelegate<TSharedPtr<IMenu>(FParseRecord& Record)> OnGetMenu;
};

template<typename ContextMenuType>
FParseResult FPhraseContextMenuNode<ContextMenuType>::ParsePhrase(TArray<FString>& InPhraseWordArray,
 FParseRecord& InParseRecord)
{
    if (!HasContextObject(InParseRecord.GetContextStack()))
    {
        UPhraseTreeContextObject* NewObject = CreateContextObject(InParseRecord);

        InParseRecord.PushContextObj(NewObject);
    }

    if (InPhraseWordArray.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Emptied Phrase Array ||"))

        return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
    }

    InPhraseWordArray.Pop();

    OnPhraseParsed.ExecuteIfBound(InParseRecord);

    return ParseChildren(InPhraseWordArray, InParseRecord);

    return FPhraseNode::ParsePhrase(InPhraseWordArray, InParseRecord);
}

template<typename ContextMenuType>
inline FParseResult FPhraseContextMenuNode<ContextMenuType>::ParsePhraseAsContext(TArray<FString>&
 InPhraseWordArray, FParseRecord& InParseRecord)
{
    if (!HasContextObject(InParseRecord.GetContextStack()))
    {
        UPhraseTreeContextObject* NewObject = CreateContextObject(InParseRecord);

        InParseRecord.PushContextObj(NewObject);
    }

    if (InPhraseWordArray.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Emptied Phrase Array ||"))

        return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
    }

    OnPhraseParsed.ExecuteIfBound(InParseRecord);

    return ParseChildren(InPhraseWordArray, InParseRecord);
}

template<typename ContextMenuType>
bool FPhraseContextMenuNode<ContextMenuType>::HasContextObject(TArray<UPhraseTreeContextObject*>
 InContextObjects) const
{
    for (auto& ContextObject : InContextObjects)
    {
        if (ContextObject->IsA(ContextMenuType::StaticClass()) && ContextObject->GetContextRoot() ==
 AsShared())
        {
            return true;
        }
    }

    return false;
}

template<typename ContextMenuType>
UPhraseTreeContextObject* FPhraseContextMenuNode<ContextMenuType>::CreateContextObject(FParseRecord&
 Record)
{
    if (!OnGetMenu.IsBound())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("OnGetMenu Delegate Not Bound. Cannot Create Context
 Object, linked to a Menu."));
        return nullptr;
    }

    TSharedPtr<IMenu> NewMenu = OnGetMenu.Execute(Record);

    if (!NewMenu.IsValid())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("OnGetMenu Delegate Returned Invalid Menu. Cannot
 Create Context Object."));
        return nullptr;
    }

    ContextMenuType* NewContextObject = NewObject<ContextMenuType>();
    NewContextObject->Init(NewMenu.ToSharedRef(), this->AsShared());

    NewContextObject->ScaleMenu(ContextMenuScalar);

    return NewContextObject;
}

template<typename ContextMenuType>
void FPhraseContextMenuNode<ContextMenuType>::ConstructContextChildren(TPhraseNodeArray& InChildNodes)
{
    // Construct Context Specific Children Nodes,
    // With Linked Functionality to the Context Menu Object and Root Node.
    TSharedPtr<FPhraseEventNode> CloseContextNode = MakeShared<FPhraseEventNode>();
    CloseContextNode->OnPhraseParsed.BindLambda(
        [this](FParseRecord& Record) {

            UPhraseTreeContextMenuObject* ContextMenu =
 Record.GetContextObj<UPhraseTreeContextMenuObject>();
            if (ContextMenu->GetContextRoot() == this->AsShared())
            {
                ContextMenu->Close();
                ContextMenu->RemoveFromRoot();

                Record.PopContextObj();
            }
        }
    );

    this->ChildNodes = TPhraseNodeArray{
        MakeShared<FPhraseNode>(TEXT("CLOSE"),
        TPhraseNodeArray {
            CloseContextNode
        })
    };

    this->ChildNodes.Append(InChildNodes);
}
```

5.84 PhraseContextNode.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "PhraseTree/PhraseNode.h"
#include "PhraseTree/IPhraseContextNode.h"
#include "PhraseTree/Containers/ContextObject.h"

#include "OpenAccessibilityComLogging.h"
#include "PhraseEventNode.h"

template<class ContextType = UPhraseTreeContextObject>
class FPhraseContextNode : public FPhraseNode, public IPhraseContextNodeBase
{
public:

    FPhraseContextNode(const TCHAR* InInputString)
        : FPhraseNode(InInputString)
    {
        static_assert(std::is_base_of<UPhraseTreeContextObject, ContextType>::value, "ContextType must be a subclass of UPhraseTreeContextObject");

        TPhraseNodeArray EmptyArray = TPhraseNodeArray();
        ConstructContextChildren(EmptyArray);
    }

    FPhraseContextNode(const TCHAR* InInputString, TPhraseNodeArray InChildNodes)
        : FPhraseNode(InInputString, InChildNodes)
    {
        static_assert(std::is_base_of<UPhraseTreeContextObject, ContextType>::value, "ContextType must be a subclass of UPhraseTreeContextObject");

        ConstructContextChildren(InChildNodes);
    }

    FPhraseContextNode(const TCHAR* InInputString, TDelegate<void(FParseRecord& Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes)
        : FPhraseNode(InInputString, InOnPhraseParsed)
    {
        static_assert(std::is_base_of<UPhraseTreeContextObject, ContextType>::value, "ContextType must be a subclass of UPhraseTreeContextObject");

        ConstructContextChildren(InChildNodes);
    }

    ~FPhraseContextNode()
    {

    }

    // FPhraseNode Implementation

    virtual FParseResult ParsePhrase(TArray<FString>& InPhraseWordArray, FParseRecord& InParseRecord) override;

    virtual FParseResult ParsePhraseAsContext(TArray<FString>& InPhraseWordArray, FParseRecord& InParseRecord) override;

    // End FPhraseNode Implementation

protected:

    // FPhraseContextNodeBase Implementation

    bool HasContextObject(TArray<UPhraseTreeContextObject*> InContextObjects) const;

    virtual UPhraseTreeContextObject* CreateContextObject(FParseRecord& Record);

    virtual void ConstructContextChildren(TPhraseNodeArray& InChildNodes);

    // End FPhraseContextNodeBase Implementation

};

template<class ContextType>
FParseResult FPhraseContextNode<ContextType>::ParsePhrase(TArray<FString>& InPhraseWordArray, FParseRecord& InParseRecord)
{
    if (!HasContextObject(InParseRecord.GetContextStack()))
    {
        UPhraseTreeContextObject* NewObject = CreateContextObject(InParseRecord);

        InParseRecord.PushContextObj(NewObject);
    }

    return FPhraseNode::ParsePhrase(InPhraseWordArray, InParseRecord);
}

template<class ContextType>
FParseResult FPhraseContextNode<ContextType>::ParsePhraseAsContext(TArray<FString>& InPhraseWordArray, FParseRecord& InParseRecord)
{
    if (!HasContextObject(InParseRecord.GetContextStack()))
    {
        UPhraseTreeContextObject* NewObject = CreateContextObject(InParseRecord);

        InParseRecord.PushContextObj(NewObject);
    }

    if (InPhraseWordArray.IsEmpty())
    {
        UE_LOG(LogOpenAccessibilityCom, Log, TEXT("|| Emptied Phrase Array ||"))

            return FParseResult(PHRASE_REQUIRES_MORE, AsShared());
    }
```

```cpp

    OnPhraseParsed.ExecuteIfBound(InParseRecord);

    // Pass
    return ParseChildren(InPhraseWordArray, InParseRecord);
}

template<class ContextType>
bool FPhraseContextNode<ContextType>::HasContextObject(TArray<UPhraseTreeContextObject*>
 InContextObjects) const
{
    for (auto& ContextObject : InContextObjects)
    {
        if (ContextObject->IsA(ContextType::StaticClass()) && ContextObject->GetContextRoot() ==
 AsShared())
        {
            return true;
        }
    }

    return false;
}


template<class ContextType>
UPhraseTreeContextObject* FPhraseContextNode<ContextType>::CreateContextObject(FParseRecord& Record)
{
    ContextType* NewContextObject = NewObject<ContextType>();
    NewContextObject->Init();
    NewContextObject->SetContextRootNode(AsShared());

    return NewContextObject;
}

template<class ContextType>
void FPhraseContextNode<ContextType>::ConstructContextChildren(TPhraseNodeArray& InChildNodes)
{
    TSharedPtr<FPhraseEventNode> CloseContextNode = MakeShared<FPhraseEventNode>();
    CloseContextNode->OnPhraseParsed.BindLambda(
        [this](FParseRecord& Record) {

            UPhraseTreeContextObject* ContextObject = Record.GetContextObj();
            if (ContextObject->GetContextRoot() == this->AsShared())
            {
                ContextObject->Close();
                ContextObject->RemoveFromRoot();

                Record.PopContextObj();
            }
        }
    );

    this->ChildNodes = TPhraseNodeArray{
        MakeShared<FPhraseNode>(TEXT("CLOSE"),
        TPhraseNodeArray {
            CloseContextNode
        })
    };

    this->ChildNodes.Append(InChildNodes);
}
```

5.85 PhraseDirectionalInputNode.h

```cpp
#pragma once

#include "CoreMinimal.h"

#include "PhraseEnumInputNode.h"
#include "Containers/Input/InputContainers.h"

class OPENACCESSIBILITYCOMMUNICATION_API FPhraseDirectionalInputNode : public
 FPhraseEnumInputNode<EPhraseDirectionalInput>
{
public:
    FPhraseDirectionalInputNode(const TCHAR* NodeName)
        : FPhraseEnumInputNode<EPhraseDirectionalInput>(NodeName)
    {}

    FPhraseDirectionalInputNode(const TCHAR* NodeName, TPhraseNodeArray InChildNodes)
        : FPhraseEnumInputNode<EPhraseDirectionalInput>(NodeName, InChildNodes)
    {}

    FPhraseDirectionalInputNode(const TCHAR* NodeName, TDelegate<void(FParseRecord& Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes)
        : FPhraseEnumInputNode<EPhraseDirectionalInput>(NodeName, InOnPhraseParsed, InChildNodes)
    {}

    FPhraseDirectionalInputNode(const TCHAR* NodeName, TPhraseNodeArray InChildNodes, TDelegate<void(int32 Input)> InOnInputRecieved)
        : FPhraseEnumInputNode<EPhraseDirectionalInput>(NodeName, InChildNodes, InOnInputRecieved)
    {}

    FPhraseDirectionalInputNode(const TCHAR* NodeName, TDelegate<void(FParseRecord& Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes, TDelegate<void(int32 Input)> InOnInputRecieved)
        : FPhraseEnumInputNode<EPhraseDirectionalInput>(NodeName, InOnPhraseParsed, InChildNodes, InOnInputRecieved)
    {}
};

class OPENACCESSIBILITYCOMMUNICATION_API FPhrase2DDirectionalInputNode : public FPhraseEnumInputNode<EPhrase2DDirectionalInput>
{
public:
    FPhrase2DDirectionalInputNode(const TCHAR* NodeName)
        : FPhraseEnumInputNode<EPhrase2DDirectionalInput>(NodeName)
    {}

    FPhrase2DDirectionalInputNode(const TCHAR* NodeName, TPhraseNodeArray InChildNodes)
        : FPhraseEnumInputNode<EPhrase2DDirectionalInput>(NodeName, InChildNodes)
    {}

    FPhrase2DDirectionalInputNode(const TCHAR* NodeName, TDelegate<void(FParseRecord& Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes)
        : FPhraseEnumInputNode<EPhrase2DDirectionalInput>(NodeName, InOnPhraseParsed, InChildNodes)
    {}

    FPhrase2DDirectionalInputNode(const TCHAR* NodeName, TPhraseNodeArray InChildNodes, TDelegate<void(int32 Input)> InOnInputRecieved)
        : FPhraseEnumInputNode<EPhrase2DDirectionalInput>(NodeName, InChildNodes, InOnInputRecieved)
    {}

    FPhrase2DDirectionalInputNode(const TCHAR* NodeName, TDelegate<void (FParseRecord& Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes, TDelegate<void(int32 Input)> InOnInputRecieved)
        : FPhraseEnumInputNode<EPhrase2DDirectionalInput>(NodeName, InOnPhraseParsed, InChildNodes, InOnInputRecieved)
    {}
};

class OPENACCESSIBILITYCOMMUNICATION_API FPhraseScrollInputNode : public FPhraseEnumInputNode<EPhraseScrollInput>
{
public:
    FPhraseScrollInputNode(const TCHAR* NodeName)
        : FPhraseEnumInputNode<EPhraseScrollInput>(NodeName)
    {}

    FPhraseScrollInputNode(const TCHAR* NodeName, TPhraseNodeArray InChildNodes)
        : FPhraseEnumInputNode<EPhraseScrollInput>(NodeName, InChildNodes)
    {}

    FPhraseScrollInputNode(const TCHAR* NodeName, TDelegate<void(FParseRecord& Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes)
        : FPhraseEnumInputNode<EPhraseScrollInput>(NodeName, InOnPhraseParsed, InChildNodes)
    {}

    FPhraseScrollInputNode(const TCHAR* NodeName, TPhraseNodeArray InChildNodes, TDelegate<void(int32 Input)> InOnInputRecieved)
        : FPhraseEnumInputNode<EPhraseScrollInput>(NodeName, InChildNodes, InOnInputRecieved)
    {}

    FPhraseScrollInputNode(const TCHAR* NodeName, TDelegate<void(FParseRecord& Record)> InOnPhraseParsed, TPhraseNodeArray InChildNodes, TDelegate<void(int32 Input)> InOnInputRecieved)
        : FPhraseEnumInputNode<EPhraseScrollInput>(NodeName, InOnPhraseParsed, InChildNodes, InOnInputRecieved)
    {}
};

class OPENACCESSIBILITYCOMMUNICATION_API FPhrasePositionalInputNode : public FPhraseEnumInputNode<EPhrasePositionalInput>
{
public:
    FPhrasePositionalInputNode(const TCHAR* NodeName)
        : FPhraseEnumInputNode<EPhrasePositionalInput>(NodeName)
    {}

    FPhrasePositionalInputNode(const TCHAR* NodeName, TPhraseNodeArray InChildNodes)
        : FPhraseEnumInputNode<EPhrasePositionalInput>(NodeName, InChildNodes)
    {}

    FPhrasePositionalInputNode(const TCHAR* NodeName, TDelegate<void(FParseRecord& Record)>
    InOnPhraseParsed, TPhraseNodeArray InChildNodes)
        : FPhraseEnumInputNode<EPhrasePositionalInput>(NodeName, InOnPhraseParsed, InChildNodes)
    {}

    FPhrasePositionalInputNode(const TCHAR* NodeName, TPhraseNodeArray InChildNodes,
    TDelegate<void(int32 Input)> InOnInputRecieved)
        : FPhraseEnumInputNode<EPhrasePositionalInput>(NodeName, InChildNodes, InOnInputRecieved)
    {}

    FPhrasePositionalInputNode(const TCHAR* NodeName, TDelegate<void(FParseRecord& Record)>
    InOnPhraseParsed, TPhraseNodeArray InChildNodes, TDelegate<void(int32 Input)> InOnInputRecieved)
        : FPhraseEnumInputNode<EPhrasePositionalInput>(NodeName, InOnPhraseParsed, InChildNodes,
    InOnInputRecieved)
    {}
};
```

5.86 PhraseEnumInputNode.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "PhraseInputNode.h"
#include "Containers/Input/InputContainers.h"

template<typename TEnum>
class OPENACCESSIBILITYCOMMUNICATION_API FPhraseEnumInputNode : public FPhraseInputNode<int32>
{
public:

    FPhraseEnumInputNode(const TCHAR* InInputString);
    FPhraseEnumInputNode(const TCHAR* InInputString, TPhraseNodeArray InChildNodes);
    FPhraseEnumInputNode(const TCHAR* InInputString, TDelegate<void(FParseRecord& Record)>
    InOnPhraseParsed, TPhraseNodeArray InChildNodes);
    FPhraseEnumInputNode(const TCHAR* InInputString, TPhraseNodeArray InChildNodes,
    TDelegate<void(int32 Input)> InOnInputRecieved);
    FPhraseEnumInputNode(const TCHAR* InInputString, TDelegate<void(FParseRecord& Record)>
    InOnPhraseParsed, TPhraseNodeArray InChildNodes, TDelegate<void(int32 Input)> InOnInputRecieved);

    ~FPhraseEnumInputNode();

protected:

    virtual bool MeetsInputRequirements(const FString& InPhrase) override;

    virtual bool RecordInput(const FString& InInput, FParseRecord& OutParseRecord) override;
};
```

5.87 PhraseEventNode.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "PhraseTree/PhraseNode.h"

class OPENACCESSIBILITYCOMMUNICATION_API FPhraseEventNode : public FPhraseNode
{
public:
    FPhraseEventNode();
    FPhraseEventNode(TDelegate<void(FParseRecord&)> InEvent);
    FPhraseEventNode(TFunction<void(FParseRecord&)> InEventFunction);

    ~FPhraseEventNode();

    // FPhraseNode Implementation
    virtual bool IsLeafNode() const override { return true; }

    virtual bool RequiresPhrase(const FString InPhrase) override;
    virtual bool RequiresPhrase(const FString InPhrase, int32& OutDistance);

    virtual FParseResult ParsePhrase(TArray<FString>& InPhraseArray, FParseRecord& InParseRecord)
    override;
    // End FPhraseNode Implementation
};
```

5.88 PhraseInputNode.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "PhraseTree/PhraseNode.h"

template <typename InputType = int32>
class OPENACCESSIBILITYCOMMUNICATION_API FPhraseInputNode : public FPhraseNode
{
public:
    FPhraseInputNode(const TCHAR* InInputString);
    FPhraseInputNode(const TCHAR* InInputString, TPhraseNodeArray InChildNodes);
    FPhraseInputNode(const TCHAR* InInputString, TDelegate<void(FParseRecord& Record)>
 InOnPhraseParsed, TPhraseNodeArray InChildNodes);
    FPhraseInputNode(const TCHAR* InInputString, TPhraseNodeArray InChildNodes, TDelegate<void
 (InputType Input)> InOnInputRecieved);
    FPhraseInputNode(const TCHAR* InInputString, TDelegate<void(FParseRecord& Record)>
 InOnPhraseParsed, TPhraseNodeArray InChildNodes, TDelegate<void(InputType Input)> InOnInputRecieved);

    ~FPhraseInputNode();

    // FPhraseNode Implementation

    virtual bool RequiresPhrase(const FString InPhrase) override;

    virtual bool RequiresPhrase(const FString InPhrase, int32& OutDistance) override;

    virtual FParseResult ParsePhrase(TArray<FString>& InPhraseArray, FParseRecord& InParseRecord)
 override;

    // End FPhraseNode Implementation

    TDelegate<void(InputType ReceivedInput)> OnInputReceived;

protected:

    virtual bool MeetsInputRequirements(const FString& InPhrase);

    virtual bool RecordInput(const FString& InInput, FParseRecord& OutParseRecord);
};
```

5.89 PhraseNode.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "PhraseTree/Containers/ParseResult.h"
#include "PhraseTree/Containers/ParseRecord.h"

class IPhraseNodeBase
{
public:

    virtual bool IsLeafNode() const = 0;

    virtual bool HasLeafChild() const = 0;

    virtual bool RequiresPhrase(const FString InPhrase) = 0;

    virtual FParseResult ParsePhrase(TArray<FString>& InPhraseWordArray, FParseRecord& InParseRecord)
 = 0;

    virtual FParseResult ParsePhraseAsContext(TArray<FString>& InPhraseWordArray, FParseRecord&
 InParseRecord) = 0;
};

class OPENACCESSIBILITYCOMMUNICATION_API FPhraseNode :  public TSharedFromThis<FPhraseNode>
{
public:

    FPhraseNode(const TCHAR* InBoundPhrase);
    FPhraseNode(const TCHAR* InBoundPhrase, TDelegate<void (FParseRecord& Record)> InOnPhraseParsed);
    FPhraseNode(const TCHAR* InBoundPhrase, TPhraseNodeArray InChildNodes);
    FPhraseNode(const TCHAR* InBoundPhrase, TDelegate<void(FParseRecord& Record)> InOnPhraseParsed,
 TPhraseNodeArray InChildNodes);

    virtual ~FPhraseNode();
    virtual bool IsLeafNode() const { return false; }

    virtual bool HasLeafChild() const;

    virtual bool RequiresPhrase(const FString InPhrase);

    virtual bool RequiresPhrase(const FString InPhrase, int32& OutDistance);

    virtual FParseResult ParsePhrase(TArray<FString>& InPhraseWordArray, FParseRecord& InParseRecord);

    virtual FParseResult ParsePhraseAsContext(TArray<FString>& InPhraseWordArray, FParseRecord&
 InParseRecord);

    virtual FParseResult ParsePhraseIfRequired(TArray<FString>& InPhraseWordArray, FParseRecord&
 InParseRecord);

    virtual FParseResult ParseChildren(TArray<FString>& InPhraseArray, FParseRecord& InParseRecord);

    bool CanBindChild(TPhraseNode& InNode);

    bool BindChildNode(TPhraseNode InNode);

    bool BindChildNodeForce(TPhraseNode InNode);

    bool BindChildrenNodes(TPhraseNodeArray InNodes);

    bool BindChildrenNodesForce(TPhraseNodeArray InNodes);

protected:

    bool HasLeafChild();

public:

    TWeakPtr<FPhraseNode> ParentNode;

    TPhraseNodeArray ChildNodes;

    FString BoundPhrase;

    // Phrase To Be Executed On the Parse Command
    TDelegate<void (FParseRecord& Record)> OnPhraseParsed;

protected:

    bool bHasLeafChild;
};
```

5.90 PhraseStringInputNode.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "PhraseInputNode.h"

class OPENACCESSIBILITYCOMMUNICATION_API FPhraseStringInputNode : public FPhraseInputNode<FString>
{
public:

    FPhraseStringInputNode(const TCHAR* InInputString);
    FPhraseStringInputNode(const TCHAR* InInputString, TPhraseNodeArray InChildNodes);
    FPhraseStringInputNode(const TCHAR* InInputString, TDelegate<void(FParseRecord& Record)>
 InOnPhraseParsed, TPhraseNodeArray InChildNodes);
    FPhraseStringInputNode(const TCHAR* InInputString, TPhraseNodeArray InChildNodes,
 TDelegate<void(FString Input)> InOnInputRecieved);
    FPhraseStringInputNode(const TCHAR* InInputString, TDelegate<void(FParseRecord& Record)>
 InOnPhraseParsed, TPhraseNodeArray InChildNodes, TDelegate<void(FString Input)> InOnInputRecieved);

    ~FPhraseStringInputNode();

protected:

    // FPhraseInputNode Implementation

    virtual bool MeetsInputRequirements(const FString& InPhrase) override;

    virtual bool RecordInput(const FString& InInput, FParseRecord& OutParseRecord) override;

    // End FPhraseInputNode Implementation
};
```

5.91 PhraseTreeFunctionLibrary.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "UObject/ObjectMacros.h"
#include "UObject/Object.h"
#include "UObject/UnrealType.h"
#include "UObject/ScriptMacros.h"

#include "PhraseTree.h"

#include "PhraseTree/Containers/ParseRecord.h"
#include "PhraseTree/Containers/Input/UParseIntInput.h"
#include "PhraseTree/Containers/Input/UParseStringInput.h"
#include "PhraseTree/Containers/Input/UParseEnumInput.h"

#include "PhraseTreeFunctionLibrary.generated.h"

// Utility Definitions

DECLARE_LOG_CATEGORY_EXTERN(LogOpenAccessibilityPhraseEvent, Log, All);

DEFINE_LOG_CATEGORY(LogOpenAccessibilityPhraseEvent);

UCLASS(Abstract)
class OPENACCESSIBILITYCOMMUNICATION_API UPhraseTreeFunctionLibrary : public UObject
{
    GENERATED_BODY()

public:

    virtual void BindBranches(TSharedRef<FPhraseTree> PhraseTree) {};

};
```

5.92 Utils.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "OpenAccessibility.h"
#include "OpenAccessibilityCommunication.h"

// Utility Macros

#define EMPTY_ARG

#define GET_ACTIVE_TAB_RETURN(ActiveContainerName, ReturnObject)                 \
  TSharedPtr<SWidget> ActiveContainerName;                                       \
  {                                                                              \
    TSharedPtr<SDockTab> _AT = FGlobalTabmanager::Get()->GetActiveTab();         \
    if (_AT == nullptr || !_AT.IsValid()) {                                      \
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display,                         \
            TEXT("GET_ACTIVE_TAB: NO ACTIVE TAB FOUND"));                        \
      return ReturnObject;                                                       \
    }                                                                            \
                                                                                 \
    ActiveContainerName = _AT->GetContent();                                     \
    if (_AT == nullptr || !ActiveContainerName.IsValid()) {                      \
        UE_LOG(LogOpenAccessibilityPhraseEvent, Display,                         \
            TEXT("GET_ACTIVE_TAB: FOUND ACTIVE TAB IS NOT VALID"));              \
        return ReturnObject;                                                     \
    }                                                                            \
  };

#define GET_ACTIVE_TAB(ActiveContainerName) \
    GET_ACTIVE_TAB_RETURN(ActiveContainerName, EMPTY_ARG)

#define GET_CAST_ACTIVE_TAB_RETURN(ActiveContainerName, ActiveTabType, ReturnObject)
           \
    static_assert(TIsDerivedFrom<ActiveTabType, SWidget>::IsDerived, "Provided Type Is Not a Valid
 Widget Type");\
    TSharedPtr<ActiveTabType> ActiveContainerName;
           \
    {
           \
        GET_ACTIVE_TAB_RETURN(_PreCastContainer, ReturnObject); \
        ActiveContainerName = StaticCastSharedPtr<ActiveTabType>(_PreCastContainer); \
        if (!ActiveContainerName.IsValid() || ActiveContainerName->GetType() != #ActiveTabType) { \
          UE_LOG(LogOpenAccessibilityPhraseEvent, Display, \
                 TEXT("GET_ACTIVE_TAB: FOUND ACTIVE TAB IS NOT VALID")); \
          return ReturnObject; \
        } \
    };

#define GET_CAST_ACTIVE_TAB(ActiveContainerName, ActiveTabType) \
    GET_CAST_ACTIVE_TAB_RETURN(ActiveContainerName, ActiveTabType, EMPTY_ARG)

#define GET_ACTIVE_KEYBOARD_WIDGET_RETURN(ActiveContainerName, ReturnObject)   \
  TSharedPtr<SWidget> ActiveContainerName;                                      \
  {                                                                             \
    FSlateApplication &SlateApp = FSlateApplication::Get();                     \
    if (!SlateApp.IsInitialized())                                              \
      return ReturnObject;                                                      \
                                                                                \
    ActiveContainerName = SlateApp.GetKeyboardFocusedWidget();                  \
    if (!ActiveContainerName.IsValid()) {                                       \
      UE_LOG(LogOpenAccessibilityPhraseEvent, Display,                          \
             TEXT("GET_ACTIVE_KEYBOARD_WIDGET: NO ACTIVE WIDGET FOUND."));      \
      return ReturnObject;                                                      \
    }                                                                           \
  };

#define GET_ACTIVE_KEYBOARD_WIDGET(ActiveContainerName) \
    GET_ACTIVE_KEYBOARD_WIDGET_RETURN(ActiveContainerName, EMPTY_ARG)

#define GET_TOP_CONTEXT_RETURN(InRecord, ContextObjectName, ContextObjectType, ReturnObject) \
  ContextObjectType *ContextObjectName;                                                     \
  {                                                                                         \
    ContextObjectName = InRecord.GetContextObj<ContextObjectType>();                        \
    if (ContextObjectName == nullptr) {                                                     \
      UE_LOG(LogOpenAccessibilityPhraseEvent, Display,                                      \
             TEXT("GET_TOP_CONTEXT: NO CONTEXT OBJECT FOUND."))                             \
      return ReturnObject;                                                                  \
    }                                                                                       \
  };

#define GET_TOP_CONTEXT(InRecord, ContextObjectName, ContextObjectType) \
    GET_TOP_CONTEXT_RETURN(InRecord, ContextObjectName, ContextObjectType, EMPTY_ARG)

// Utility Functions

FORCEINLINE TSharedRef<FPhraseTree> GetPhraseTree()
{
    FOpenAccessibilityCommunicationModule &OAComsModule = FOpenAccessibilityCommunicationModule::Get();

    if (OAComsModule.PhraseTree.IsValid())
        return OAComsModule.PhraseTree.ToSharedRef();

    return TSharedRef<FPhraseTree>();
}

FORCEINLINE TSharedRef<FAssetAccessibilityRegistry> GetAssetRegistry()
{
    FOpenAccessibilityModule &OAModule = FOpenAccessibilityModule::Get();

    if (OAModule.AssetAccessibilityRegistry.IsValid())
        return OAModule.AssetAccessibilityRegistry.ToSharedRef();

    return TSharedRef<FAssetAccessibilityRegistry>();
}

// Delegate Utilities

template<typename ObjectType>
[[nodiscard]] FORCEINLINE TDelegate<void(FParseRecord&)> CreateParseDelegate(ObjectType* ObjPtr, void (ObjectType::* ObjFunction)(FParseRecord&))
{
    return TDelegate<void(FParseRecord&)>::CreateUObject(ObjPtr, ObjFunction);
}

template <typename ObjectType, typename InputType>
[[nodiscard]] FORCEINLINE TDelegate<void(InputType)> CreateInputDelegate(ObjectType* ObjPtr, void (ObjectType::* ObjFunction)(InputType))
{
    return TDelegate<void(InputType)>::CreateUObject(ObjPtr, ObjFunction);
}

template <typename ObjectType>
[[nodiscard]] FORCEINLINE TDelegate<TSharedPtr<IMenu>(FParseRecord&)> CreateMenuDelegate(ObjectType*
 ObjPtr, TSharedPtr<IMenu> (ObjectType::* ObjFunction)(FParseRecord&))
{
    return TDelegate<TSharedPtr<IMenu>(FParseRecord&)>::CreateUObject(ObjPtr, ObjFunction);
}


// Utility Functions
namespace EventUtils
{
    [[nodiscard]] FORCEINLINE FString RemoveWordsFromEnd(const FString& InString, const int32&
 AmountToRemove)
    {
        TArray<FString> SplitTextBoxString;
        InString.ParseIntoArrayWS(SplitTextBoxString);

        int RemovedAmount = 0;
        int CurrentIndex = SplitTextBoxString.Num() - 1;
        while (RemovedAmount < AmountToRemove) {
            if (SplitTextBoxString.IsEmpty())
                break;

            SplitTextBoxString.RemoveAt(CurrentIndex--);
            RemovedAmount++;
        }

        if (SplitTextBoxString.Num() > 0)
            return FString::Join(SplitTextBoxString, TEXT(" "));

        return TEXT("");
    }
}
```

5.93 Utils.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

class OPENACCESSIBILITYCOMMUNICATION_API NumericParser
{
public:

    static bool IsValidNumeric(const FString& StringToCheck, bool ConvertToUpper = true);

    static void StringToNumeric(FString& NumericString, bool ConvertToUpper = true);

private:
    static const TMap<const FString, const FString> StringMappings;
};
```

5.94 PhraseTreeUtils.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#include "PhraseTree/PhraseTreeFunctionLibrary.h"

#include "PhraseTreeUtils.generated.h"

UCLASS()
class OPENACCESSIBILITYCOMMUNICATION_API UPhraseTreeUtils : public UObject
{
    GENERATED_BODY()

public:

    UPhraseTreeUtils();

    virtual ~UPhraseTreeUtils();

    // Function Library Methods

    void RegisterFunctionLibrary(UPhraseTreeFunctionLibrary* LibraryToRegister);

    void SetPhraseTree(TSharedRef<FPhraseTree> NewPhraseTree)
    {
        this->PhraseTree = NewPhraseTree;
    }

protected:

    UPROPERTY(EditAnywhere)
    TArray<UPhraseTreeFunctionLibrary*> RegisteredLibraries;


    TWeakPtr<FPhraseTree> PhraseTree;
};

```

5.95 SocketCommunicationServer.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"

#ifdef WITH_ZEROMQ
#include "zmq.hpp"
#include "zmq_addon.hpp"
#else
#error "ZeroMQ Could not be found. Please Make Sure ZEROMQ is Installed Correctly, and the WITH_ZEROMQ
 Definition is Valid."
#endif // WITH_ZEROMQ

class FJsonObject;

typedef zmq::send_flags ComSendFlags;
typedef zmq::recv_flags ComRecvFlags;

class OPENACCESSIBILITYCOMMUNICATION_API FSocketCommunicationServer
{
public:

    FSocketCommunicationServer(const std::string SendAddress = "tcp://127.0.0.1:5555", const
 std::string RecvAddress = "tcp://127.0.0.1:5556", const int PollTimeout = 10);
    ~FSocketCommunicationServer();

    bool EventOccured();

    bool SendArrayBuffer(const float* MessageData, size_t Size, ComSendFlags SendFlags =
 ComSendFlags::none);

    bool SendArrayBuffer(const float MessageData[], ComSendFlags SendFlags = ComSendFlags::none);

    bool SendArrayBuffer(const TArray<float>& ArrayMessage, ComSendFlags SendFlags =
 ComSendFlags::none);

    bool SendArrayMessage(const float* MessageData, size_t Size, ComSendFlags SendFlags =
 ComSendFlags::none);

    bool SendArrayMessage(const float MessageData[], ComSendFlags SendFlags = ComSendFlags::none);

    bool SendArrayMessage(const TArray<float>& ArrayMessage, ComSendFlags SendFlags =
 ComSendFlags::none);

    bool SendArrayMessageWithMeta(const float* MessageData, size_t Size, const
 TSharedRef<FJsonObject>& Metadata, ComSendFlags SendFlags = ComSendFlags::none);

    bool SendArrayMessageWithMeta(const float MessageData[], const TSharedRef<FJsonObject>& Metadata,
 ComSendFlags SendFlags = ComSendFlags::none);

    bool SendArrayMessageWithMeta(const TArray<float>& ArrayMessage, const TSharedRef<FJsonObject>&
 Metadata, ComSendFlags SendFlags = ComSendFlags::none);

    bool SendStringBuffer(const std::string StringMessage, ComSendFlags SendFlags =
 ComSendFlags::none);

    bool SendJsonBuffer(const std::string JsonMessage, ComSendFlags SendFlags = ComSendFlags::none);

    template <typename T>
    bool RecvArray(TArray<T>& OutArrayData, size_t Size, ComRecvFlags RecvFlag = ComRecvFlags::none);

    bool RecvString(FString& OutStringMessage, ComRecvFlags RecvFlag = ComRecvFlags::none);

    bool RecvJson(FString& OutJsonMessage, ComRecvFlags RecvFlag = ComRecvFlags::none);

    bool RecvStringMultipart(TArray<FString>& OutMessages, ComRecvFlags RecvFlag =
 ComRecvFlags::none);

    bool RecvStringMultipartWithMeta(TArray<FString>& OutMessages, TSharedPtr<FJsonObject>&
 OutMetadata, ComRecvFlags RecvFlag = ComRecvFlags::none);

protected:

    bool RecvMultipartWithMeta(std::vector<zmq::message_t>& OutMultipartMessages,
 TSharedPtr<FJsonObject>& OutMetadata, ComRecvFlags RecvFlags);

    bool SerializeJSON(const TSharedRef<FJsonObject>& InJsonObject, FString& OutJsonString);

    bool DeserializeJSON(const FString& InJsonString, TSharedPtr<FJsonObject>& OutJsonObject);

protected:

    zmq::context_t* Context;

    zmq::socket_t* SendSocket;

    zmq::socket_t* RecvSocket;

    zmq::poller_t<int>* Poller;

    std::string SendAddress;
    std::string RecvAddress;

    int PollTimeout;
};
```

5.96 UBAudioCapture.h

```
// Copyright F-Dudley. All Rights Reserved.

#pragma once

#include "CoreMinimal.h"
#include "AudioCapture.h"

class OPENACCESSIBILITYCOMMUNICATION_API UBAudioCapture : public UAudioCapture
{

public:
    UBAudioCapture();
    ~UBAudioCapture();

    bool OpenDefaultAudioStream(int32 OverrideSampleRate, int32 OverrideInputChannels);
};
```

Index

__del__
 OpenAccessibilityPy.Audio.AudioResampler, 9
 OpenAccessibilityPy.CommunicationServer.CommunicationServer,
 12
 OpenAccessibilityPy.OpenAccessibilityPy, 183
 OpenAccessibilityPy.WhisperInterface.WhisperInterface,
 308
__init__
 OpenAccessibilityPy.Audio.AudioResampler, 9
 OpenAccessibilityPy.CommunicationServer.CommunicationServer,
 12
 OpenAccessibilityPy.OpenAccessibilityPy, 182
 OpenAccessibilityPy.WhisperInterface.WhisperInterface,
 308
~FAccessibilityNodeFactory
 FAccessibilityNodeFactory, 21
~FAssetAccessibilityRegistry
 FAssetAccessibilityRegistry, 26
~FGraphIndexer
 FGraphIndexer, 36
~FIndexer
 FIndexer< KeyType, ValueType >, 50
~FParseRecord
 FParseRecord, 74
~FPhraseContextMenuNode
 FPhraseContextMenuNode< ContextMenuType >,
 91
~FPhraseContextNode
 FPhraseContextNode< ContextType >, 97
~FPhraseEnumInputNode
 FPhraseEnumInputNode< TEnum >, 104
~FPhraseEventNode
 FPhraseEventNode, 107
~FPhraseInputNode
 FPhraseInputNode< InputType >, 112
~FPhraseNode
 FPhraseNode, 118
~FPhraseStringInputNode
 FPhraseStringInputNode, 132
~FPhraseTree
 FPhraseTree, 135
~FPhraseTreeBranchBind
 FPhraseTreeBranchBind, 140
~FPhraseTreeContextManager
 FPhraseTreeContextManager, 142
~FSocketCommunicationServer
 FSocketCommunicationServer, 147
~FTranscriptionVisualizer
 FTranscriptionVisualizer, 164
~OAEditorAccessibilityManager
 OAEditorAccessibilityManager, 177
~SAccessibilityTranscriptionVis
 SAccessibilityTranscriptionVis, 187
~SContentIndexer
 SContentIndexer, 191
~SIndexer
 SIndexer, 198
~TGraphAccessibilityNodeFactory
 TGraphAccessibilityNodeFactory< T >, 202
~UAccessibilityAddNodeContextMenu
 UAccessibilityAddNodeContextMenu, 209
~UAccessibilityGraphLocomotionContext
 UAccessibilityGraphLocomotionContext, 235
~UAccessibilityWindowToolbar
 UAccessibilityWindowToolbar, 245
~UAudioManager
 UAudioManager, 248
~UBAudioCapture
 UBAudioCapture, 253
~ULocalizedInputLibrary
 ULocalizedInputLibrary, 255
~UNodeInteractionLibrary
 UNodeInteractionLibrary, 261
~UParseEnumInput
 UParseEnumInput, 280
~UParseInput
 UParseInput, 282
~UParseIntInput
 UParseIntInput, 283
~UParseStringInput
 UParseStringInput, 285
~UPhraseTreeContextMenuObject
 UPhraseTreeContextMenuObject, 288
~UPhraseTreeContextObject
 UPhraseTreeContextObject, 294
~UPhraseTreeUtils
 UPhraseTreeUtils, 297
~UViewInteractionLibrary
 UViewInteractionLibrary, 300
~UWindowInteractionLibrary
 UWindowInteractionLibrary, 305

AccessibilityNodeFactory
 FOpenAccessibilityModule, 69
AccessibilityRegistry
 TGraphAccessibilityNodeFactory< T >, 205
AddNode
 FGraphIndexer, 36, 37
AddPhraseInput

FParseRecord, 74
AddPhraseString
FParseRecord, 74
AddValue
FIndexer< KeyType, ValueType >, 51
AppendFilterText
UAccessibilityAddNodeContextMenu, 209
UAccessibilityGraphEditorContext, 222
AppendScrollDistance
UAccessibilityAddNodeContextMenu, 209
UAccessibilityGraphEditorContext, 223
ApplyAccessibilityWidget
UAccessibilityAddNodeContextMenu, 210
AssetAccessibilityRegistry
FOpenAccessibilityModule, 69
audio_resampler
OpenAccessibilityPy.OpenAccessibilityPy, 185
AudioManager
FOpenAccessibilityCommunicationModule, 66
AvailableIndexes
FIndexer< KeyType, ValueType >, 59
AvailableIndices
FGraphIndexer, 45

beam_size
OpenAccessibilityPy.WhisperInterface.WhisperInterface, 310
bHasLeafChild
FPhraseNode, 126
BindBranch
FPhraseTree, 135
BindBranches
FPhraseTree, 136
ULocalizedInputLibrary, 255
UNodeInteractionLibrary, 261
UPhraseTreeFunctionLibrary, 296
UViewInteractionLibrary, 301
UWindowInteractionLibrary, 305
BindChildNode
FPhraseNode, 119
BindChildNodeForce
FPhraseNode, 119
BindChildrenNodes
FPhraseNode, 120
BindChildrenNodesForce
FPhraseNode, 120
BindFocusChangedEvent
UAccessibilityGraphLocomotionContext, 235
BindMenuDismissed
UPhraseTreeContextMenuObject, 288
BindTickDelegate
UPhraseTreeContextMenuObject, 289
bIsActive
UPhraseTreeContextObject, 295
BlueprintCompile
UNodeInteractionLibrary, 265
BotRight
FPanelViewPosition, 71
BottomRight
FGraphLocomotionChunk, 48
BoundPhrase
FPhraseNode, 126
BranchRoot
FPhraseTreeBranchBind, 140

CalculateVisualChunksBounds
UAccessibilityGraphLocomotionContext, 235
CanBindChild
FPhraseNode, 121
CancelLocomotion
UAccessibilityGraphLocomotionContext, 236
ChangeChunkVis
UAccessibilityGraphLocomotionContext, 236
CheckBoxes
UAccessibilityGraphEditorContext, 232
ChildNodes
FPhraseNode, 126
ChunkArray
UAccessibilityGraphLocomotionContext, 242
ChunkIndexer
FGraphLocomotionChunk, 48
ChunkSize
UAccessibilityGraphLocomotionContext, 242
ChunkVisWidget
FGraphLocomotionChunk, 48
ChunkWidget
FGraphLocomotionChunk, 49
Close
UAccessibilityAddNodeContextMenu, 210
UAccessibilityGraphEditorContext, 223
UAccessibilityGraphLocomotionContext, 236
UPhraseTreeContextMenuObject, 289
UPhraseTreeContextObject, 294
CloseActiveWindow
UWindowInteractionLibrary, 306
com_server
OpenAccessibilityPy.OpenAccessibilityPy, 185
ConfirmSelection
UAccessibilityGraphLocomotionContext, 237
Construct
SAccessibilityTranscriptionVis, 187
SContentIndexer, 191
SIndexer, 199
ConstructBottomIndexer
SContentIndexer, 191
ConstructContentContainer
SContentIndexer, 192
ConstructContextChildren
FPhraseContextMenuNode< ContextMenuType >, 91
FPhraseContextNode< ContextType >, 97
IPhraseContextNodeBase, 171
ConstructIndexContainer
SContentIndexer, 193
ConstructIndexText
SContentIndexer, 193
ConstructLeftIndexer
SContentIndexer, 193

ConstructRightIndexer
SContentIndexer, 194
ConstructTopIndexer
SContentIndexer, 195
ConstructVisualizer
FTranscriptionVisualizer, 164
ContainsKey
FGraphIndexer, 37
FIndexer< KeyType, ValueType >, 52
ContainsNode
FGraphIndexer, 37, 38
ContainsValue
FIndexer< KeyType, ValueType >, 52
Content/Python/init_unreal.py, 311
Content/Python/old_init_unreal.py, 312
Content/Python/OpenAccessibilityPy/__init__.py, 313
Content/Python/OpenAccessibilityPy/__main__.py, 315
Content/Python/OpenAccessibilityPy/Audio.py, 316
Content/Python/OpenAccessibilityPy/CommunicationServer.py, 318
Content/Python/OpenAccessibilityPy/LibUtils.py, 322
Content/Python/OpenAccessibilityPy/Logging.py, 322
Content/Python/OpenAccessibilityPy/WhisperInterface.py, 323
Content/Python/TestWhisper.py, 324
Context
FSocketCommunicationServer, 161
context
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 20
ContextAwarenessCheckBox
UAccessibilityAddNodeContextMenu, 219
ContextMenuScalar
FPhraseContextMenuNode< ContextMenuType >, 94
ContextObjectStack
FParseRecord, 83
ContextRoot
UPhraseTreeContextObject, 295
CreateAccessibilityWrapper
UAccessibilityGraphEditorContext, 223
CreateContextObject
FPhraseContextMenuNode< ContextMenuType >, 92
FPhraseContextNode< ContextType >, 97
IPhraseContextNodeBase, 171
CreateNode
FAccessibilityNodeFactory, 22
CreateNodeWidget
TGraphAccessibilityNodeFactory< T >, 203
CreatePinWidget
TGraphAccessibilityNodeFactory< T >, 204
CreateVisualGrid
UAccessibilityGraphLocomotionContext, 237
CurrentViewPosition
UAccessibilityGraphLocomotionContext, 242

DeleteNode
UNodeInteractionLibrary, 266
DeserializeJSON
FSocketCommunicationServer, 148
DoesItemsRequireRefresh
UAccessibilityAddNodeContextMenu, 211
DumpTick
FOpenAccessibilityAnalyticsModule, 60

Empty
FIndexer< KeyType, ValueType >, 53
EnumType
UParseEnumInput, 281
ERROR
OpenAccessibilityPy.Logging.LogLevel, 174
EventOccured
FSocketCommunicationServer, 148
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 13

FAccessibilityNodeFactory, 21
~FAccessibilityNodeFactory, 21
CreateNode, 22
FAccessibilityNodeFactory, 21
SetSharedPtr, 22
WrapNodeWidget, 23
WrapPinWidget, 24
FAssetAccessibilityRegistry, 25
~FAssetAccessibilityRegistry, 26
FAssetAccessibilityRegistry, 26
GetAllGraphIndexes, 26, 27
GetAllGraphKeyIndexes, 27
GetGraphIndexer, 28
GraphAssetIndex, 32
IsGameWorldAssetRegistered, 28
IsGraphAssetRegistered, 29
RegisterGameWorldAsset, 29
RegisterGraphAsset, 29, 30
UnregisterGameWorldAsset, 30
UnregisterGraphAsset, 32
FAudioManagerSettings, 33
FAudioManagerSettings, 33
LevelThreshold, 33
SaveName, 34
SavePath, 34
FGraphIndexer, 34
~FGraphIndexer, 36
AddNode, 36, 37
AvailableIndices, 45
ContainsKey, 37
ContainsNode, 37, 38
FGraphIndexer, 35
GetKey, 38, 40
GetNode, 40, 41
GetOrAddNode, 41, 42
GetPin, 42, 43
IndexMap, 45
LinkedGraph, 46
NodeSet, 46
OnGraphChangedHandle, 46
OnGraphEvent, 43

OnGraphRebuild, 44
RemoveNode, 44, 45
FGraphLocomotionChunk, 46
BottomRight, 48
ChunkIndexer, 48
ChunkVisWidget, 48
ChunkWidget, 49
GetChunkBottomRight, 47
GetChunkBounds, 47
GetChunkTopLeft, 47
SetChunkBounds, 47
SetVisColor, 48
TopLeft, 49
FilterTextBox
UAccessibilityAddNodeContextMenu, 219
UAccessibilityGraphEditorContext, 233
FIndexer
FIndexer< KeyType, ValueType >, 50
FIndexer< KeyType, ValueType >, 49
∼FIndexer, 50
AddValue, 51
AvailableIndexes, 59
ContainsKey, 52
ContainsValue, 52
Empty, 53
FIndexer, 50
GetAvailableKey, 53
GetKey, 54
GetKeyOrAddValue, 55
GetValue, 56
IndexMap, 59
IsEmpty, 57
Num, 57
RemoveValue, 58
Reset, 59
FindGraphActionMenu
UAccessibilityGraphEditorContext, 224
FindStaticComponents
UAccessibilityGraphEditorContext, 224
FindTreeView
UAccessibilityGraphEditorContext, 225
FOpenAccessibilityAnalyticsModule, 60
DumpTick, 60
Get, 61
LogEvent, 61
ShutdownModule, 61
StartupModule, 62
SupportsDynamicReloading, 62
FOpenAccessibilityCommunicationModule, 62
AudioManager, 66
Get, 63
HandleKeyDownEvent, 63
OnTranscriptionRecieved, 66
PhraseTree, 66
PhraseTreeUtils, 67
ShutdownModule, 64
SocketServer, 67
StartupModule, 64
SupportsDynamicReloading, 65
Tick, 65
TranscribeWaveForm, 65
FOpenAccessibilityModule, 67
AccessibilityNodeFactory, 69
AssetAccessibilityRegistry, 69
Get, 68
ShutdownModule, 68
StartupModule, 68
SupportsDynamicReloading, 69
FPanelViewPosition, 70
BotRight, 71
FPanelViewPosition, 70
operator!=, 71
TopLeft, 71
FParseRecord, 72
∼FParseRecord, 74
AddPhraseInput, 74
AddPhraseString, 74
ContextObjectStack, 83
FParseRecord, 73
FPhraseTree, 83
GetContextObj, 75, 76
GetContextStack, 76
GetPhraseInput, 77, 78
GetPhraseInputs, 79
GetPhraseString, 80
HasContextObj, 80
PhraseInputs, 83
PhraseRecord, 83
PopContextObj, 81
PushContextObj, 82
RemoveContextObj, 82
RemovePhraseInput, 82
FParseResult, 84
FParseResult, 84, 85
ReachedNode, 85
Result, 85
FPhrase2DDirectionalInputNode, 86
FPhrase2DDirectionalInputNode, 86, 87
FPhraseContextMenuNode
FPhraseContextMenuNode< ContextMenuType >, 89, 90
FPhraseContextMenuNode< ContextMenuType >, 88
∼FPhraseContextMenuNode, 91
ConstructContextChildren, 91
ContextMenuScalar, 94
CreateContextObject, 92
FPhraseContextMenuNode, 89, 90
HasContextObject, 92
OnGetMenu, 95
ParsePhrase, 93
ParsePhraseAsContext, 94
FPhraseContextNode
FPhraseContextNode< ContextType >, 96
FPhraseContextNode< ContextType >, 95
∼FPhraseContextNode, 97
ConstructContextChildren, 97

CreateContextObject, 97
FPhraseContextNode, 96
HasContextObject, 98
ParsePhrase, 98
ParsePhraseAsContext, 99
FPhraseDirectionalInputNode, 100
FPhraseDirectionalInputNode, 101
FPhraseEnumInputNode
FPhraseEnumInputNode< TEnum >, 103, 104
FPhraseEnumInputNode< TEnum >, 102
∼FPhraseEnumInputNode, 104
FPhraseEnumInputNode, 103, 104
MeetsInputRequirements, 104
RecordInput, 105
FPhraseEventNode, 106
∼FPhraseEventNode, 107
FPhraseEventNode, 106, 107
IsLeafNode, 107
ParsePhrase, 108
RequiresPhrase, 108, 109
FPhraseInputNode
FPhraseInputNode< InputType >, 111, 112
FPhraseInputNode< InputType >, 109
∼FPhraseInputNode, 112
FPhraseInputNode, 111, 112
MeetsInputRequirements, 112
OnInputReceived, 115
ParsePhrase, 113
RecordInput, 114
RequiresPhrase, 114, 115
FPhraseNode, 116
∼FPhraseNode, 118
bHasLeafChild, 126
BindChildNode, 119
BindChildNodeForce, 119
BindChildrenNodes, 120
BindChildrenNodesForce, 120
BoundPhrase, 126
CanBindChild, 121
ChildNodes, 126
FPhraseNode, 117, 118
HasLeafChild, 121
IsLeafNode, 122
OnPhraseParsed, 126
ParentNode, 126
ParseChildren, 122
ParsePhrase, 123
ParsePhraseAsContext, 124
ParsePhraseIfRequired, 124
RequiresPhrase, 124, 125
FPhrasePositionalInputNode, 127
FPhrasePositionalInputNode, 127, 128
FPhraseScrollInputNode, 129
FPhraseScrollInputNode, 129, 130
FPhraseStringInputNode, 131
∼FPhraseStringInputNode, 132
FPhraseStringInputNode, 131, 132
MeetsInputRequirements, 133
RecordInput, 133
FPhraseTree, 134
∼FPhraseTree, 135
BindBranch, 135
BindBranches, 136
FParseRecord, 83
FPhraseTree, 135
FPhraseTreeContextManager, 145
GetContextManager, 136
ParsePhrase, 136
ParseTranscription, 137
Tick, 139
FPhraseTreeBranchBind, 139
∼FPhraseTreeBranchBind, 140
BranchRoot, 140
FPhraseTreeBranchBind, 140
StartNode, 141
FPhraseTreeContextManager, 141
∼FPhraseTreeContextManager, 142
FPhraseTree, 145
FPhraseTreeContextManager, 142
GetContextStack, 142
HasContextObject, 142
HasContextObjects, 143
IsEmpty, 143
PeekContextObject, 143
PopContextObject, 144
PushContextObject, 145
FSocketCommunicationServer, 145
∼FSocketCommunicationServer, 147
Context, 161
DeserializeJSON, 148
EventOccured, 148
FSocketCommunicationServer, 147
Poller, 161
PollTimeout, 161
RecvAddress, 162
RecvArray, 149
RecvJson, 150
RecvMultipartWithMeta, 150
RecvSocket, 162
RecvString, 151
RecvStringMultipart, 152
RecvStringMultipartWithMeta, 152
SendAddress, 162
SendArrayBuffer, 153, 154
SendArrayMessage, 155, 156
SendArrayMessageWithMeta, 157, 158
SendJsonBuffer, 159
SendSocket, 162
SendStringBuffer, 160
SerializeJSON, 161
FTranscriptionVisualizer, 163
∼FTranscriptionVisualizer, 164
ConstructVisualizer, 164
FTranscriptionVisualizer, 163
GetDisplayVisualizerPosition, 164
GetTopScreenVisualizerPosition, 165

MoveVisualizer, 165
OnTranscriptionRecieved, 166
RegisterTicker, 166
ReparentWindow, 166
Tick, 167
TickDelegateHandle, 168
UnregisterTicker, 167
UpdateVisualizer, 167
VisContent, 168
VisWindow, 168
FTreeViewTickRequirements
UAccessibilityGraphEditorContext::FTreeViewTickRequirements, 169

GenerateVisualChunks
UAccessibilityGraphLocomotionContext, 237
Get
FOpenAccessibilityAnalyticsModule, 61
FOpenAccessibilityCommunicationModule, 63
FOpenAccessibilityModule, 68
GetActiveToolkitWidget
UAccessibilityWindowToolbar, 245
GetAllGraphIndexes
FAssetAccessibilityRegistry, 26, 27
GetAllGraphKeyIndexes
FAssetAccessibilityRegistry, 27
GetAudioCaptureNumChannels
UAudioManager, 249
GetAudioCaptureSampleRate
UAudioManager, 249
GetAvailableKey
FIndexer< KeyType, ValueType >, 53
GetChunkBottomRight
FGraphLocomotionChunk, 47
GetChunkBounds
FGraphLocomotionChunk, 47
GetChunkTopLeft
FGraphLocomotionChunk, 47
GetContent
SContentIndexer, 195, 196
GetContextManager
FPhraseTree, 136
GetContextObj
FParseRecord, 75, 76
GetContextRoot
UPhraseTreeContextObject, 294
GetContextStack
FParseRecord, 76
FPhraseTreeContextManager, 142
GetDisplayVisualizerPosition
FTranscriptionVisualizer, 164
GetEnumType
UParseEnumInput, 280
GetFilterText
UAccessibilityAddNodeContextMenu, 211
UAccessibilityGraphEditorContext, 226
GetGraphActionFromIndex
UAccessibilityAddNodeContextMenu, 211, 212
GetGraphActionFromIndexSP
UAccessibilityAddNodeContextMenu, 212
GetGraphIndexer
FAssetAccessibilityRegistry, 28
GetIndexText
SIndexer, 199
GetIsActive
UPhraseTreeContextObject, 294
GetKey
FGraphIndexer, 38, 40
FIndexer< KeyType, ValueType >, 54
GetKeyOrAddValue
FIndexer< KeyType, ValueType >, 55
GetNode
FGraphIndexer, 40, 41
GetOrAddNode
FGraphIndexer, 41, 42
GetPhraseInput
FParseRecord, 77, 78
GetPhraseInputs
FParseRecord, 79
GetPhraseString
FParseRecord, 80
GetPin
FGraphIndexer, 42, 43
GetStaticIndexOffset
UAccessibilityGraphEditorContext, 226
GetTopScreenVisualizerPosition
FTranscriptionVisualizer, 165
GetTreeViewAction
UAccessibilityGraphEditorContext, 226
GetValue
FIndexer< KeyType, ValueType >, 56
UParseIntInput, 283
UParseStringInput, 285
GetWindow
UPhraseTreeContextMenuObject, 289
GraphAssetIndex
FAssetAccessibilityRegistry, 32
GraphMenu
UAccessibilityAddNodeContextMenu, 219
UAccessibilityGraphEditorContext, 233
GridContainer
UAccessibilityGraphLocomotionContext, 243
GridParent
UAccessibilityGraphLocomotionContext, 243

HandleKeyDownEvent
FOpenAccessibilityCommunicationModule, 63
HandleTranscriptionRequest
OpenAccessibilityPy.OpenAccessibilityPy, 183
HasContextObj
FParseRecord, 80
HasContextObject
FPhraseContextMenuNode< ContextMenuType >, 92
FPhraseContextNode< ContextType >, 98
FPhraseTreeContextManager, 142
IPhraseContextNodeBase, 171
HasContextObjects

FPhraseTreeContextManager, 143
HasLeafChild
FPhraseNode, 121
IPhraseNodeBase, 172
HideNativeVisuals
UAccessibilityGraphLocomotionContext, 238

Implementation
TGraphAccessibilityNodeFactory< T >, 205
IndexedContent
SContentIndexer, 197
IndexerWidget
SContentIndexer, 197
IndexFocus
UViewInteractionLibrary, 302
IndexMap
FGraphIndexer, 45
FIndexer< KeyType, ValueType >, 59
IndexTextBlock
SIndexer, 201
INFO
OpenAccessibilityPy.Logging.LogLevel, 175
Init
UAccessibilityAddNodeContextMenu, 213, 214
UAccessibilityGraphEditorContext, 227
UAccessibilityGraphLocomotionContext, 238, 239
UPhraseTreeContextMenuObject, 289, 290
IPhraseContextNodeBase, 170
ConstructContextChildren, 171
CreateContextObject, 171
HasContextObject, 171
IPhraseNodeBase, 172
HasLeafChild, 172
IsLeafNode, 172
ParsePhrase, 173
ParsePhraseAsContext, 173
RequiresPhrase, 173
IsActiveToolbar
UAccessibilityWindowToolbar, 245
IsCapturingAudio
UAudioManager, 249
IsEmpty
FIndexer< KeyType, ValueType >, 57
FPhraseTreeContextManager, 143
IsGameWorldAssetRegistered
FAssetAccessibilityRegistry, 28
IsGraphAssetRegistered
FAssetAccessibilityRegistry, 29
IsLeafNode
FPhraseEventNode, 107
FPhraseNode, 122
IPhraseNodeBase, 172
IsValidNumeric
NumericParser, 176

KeyboardInputAdd
ULocalizedInputLibrary, 256
KeyboardInputConfirm
ULocalizedInputLibrary, 257
KeyboardInputExit
ULocalizedInputLibrary, 258
KeyboardInputRemove
ULocalizedInputLibrary, 258
KeyboardInputReset
ULocalizedInputLibrary, 259

LevelThreshold
FAudioManagerSettings, 33
LinkedEditor
UAccessibilityGraphLocomotionContext, 243
LinkedGraph
FGraphIndexer, 46
LocomotionCancel
UNodeInteractionLibrary, 266
LocomotionConfirm
UNodeInteractionLibrary, 267
LocomotionRevert
UNodeInteractionLibrary, 267
LocomotionSelect
UNodeInteractionLibrary, 267
LogEvent
FOpenAccessibilityAnalyticsModule, 61

MeetsInputRequirements
FPhraseEnumInputNode< TEnum >, 104
FPhraseInputNode< InputType >, 112
FPhraseStringInputNode, 133
Menu
UPhraseTreeContextMenuObject, 292
MoveNode
UNodeInteractionLibrary, 268
MoveViewport
UAccessibilityGraphLocomotionContext, 239
UViewInteractionLibrary, 302
MoveVisualizer
FTranscriptionVisualizer, 165

NodeAddMenu
UNodeInteractionLibrary, 269
NodeAddPinMenu
UNodeInteractionLibrary, 270
NodeAddScroll
UNodeInteractionLibrary, 271
NodeAddSearchAdd
UNodeInteractionLibrary, 272
NodeAddSearchRemove
UNodeInteractionLibrary, 273
NodeAddSearchReset
UNodeInteractionLibrary, 273
NodeAddSelect
UNodeInteractionLibrary, 273
NodeIndexFocus
UNodeInteractionLibrary, 274
NodeSet
FGraphIndexer, 46
Num
FIndexer< KeyType, ValueType >, 57
NumericParser, 175

IsValidNumeric, 176
StringToNumeric, 176

OAEditorAccessibilityManager, 177
∼OAEditorAccessibilityManager, 177
OAEditorAccessibilityManager, 177
OnAudioReadyForTranscription
UAudioManager, 252
OnDefaultDeviceChanged
UAudioManager, 250
OnFocusChanged
UAccessibilityGraphLocomotionContext, 240
OnGetMenu
FPhraseContextMenuNode< ContextMenuType >, 95
OnGraphChangedHandle
FGraphIndexer, 46
OnGraphEvent
FGraphIndexer, 43
OnGraphRebuild
FGraphIndexer, 44
OnInputReceived
FPhraseInputNode< InputType >, 115
OnMenuDismissed
UPhraseTreeContextMenuObject, 290
OnPhraseParsed
FPhraseNode, 126
OnTranscriptionRecieved
FOpenAccessibilityCommunicationModule, 66
FTranscriptionVisualizer, 166
OpenAccessibility, 177
OpenAccessibility, 178
OpenAccessibilityAnalytics, 179
OpenAccessibilityAnalytics, 179
OpenAccessibilityCommunication, 180
OpenAccessibilityCommunication, 181
OpenAccessibilityPy.Audio.AudioResampler, 9
__del__, 9
__init__, 9
resample, 10
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 11
__del__, 12
__init__, 12
context, 20
EventOccured, 13
poller, 20
poller_timeout_time, 20
ReceiveJSON, 13
ReceiveMultipart, 14
ReceiveNDArray, 14
ReceiveNDArrayWithMeta, 15
ReceiveString, 15
RecieveRaw, 16
recv_socket, 20
recv_socket_context, 20
send_socket_context, 20
SendJSON, 16
SendMultipart, 17
SendMultipartWithMeta, 17
SendNDArray, 18
SendNDArrayWithMeta, 18
SendString, 19
OpenAccessibilityPy.Logging.LogLevel, 174
ERROR, 174
INFO, 175
WARNING, 175
OpenAccessibilityPy.OpenAccessibilityPy, 182
__del__, 183
__init__, 182
audio_resampler, 185
com_server, 185
HandleTranscriptionRequest, 183
pyshutdown_handle, 186
Shutdown, 184
Tick, 185
tick_handle, 186
whisper_interface, 186
worker_pool, 186
OpenAccessibilityPy.WhisperInterface.WhisperInterface, 307
__del__, 308
__init__, 308
beam_size, 310
process_audio_buffer, 309
process_file_from_dir, 309
whisper_model, 310
OpenDefaultAudioStream
UBAudioCapture, 253
operator!=
FPanelViewPosition, 71

ParentNode
FPhraseNode, 126
ParseChildren
FPhraseNode, 122
ParsePhrase
FPhraseContextMenuNode< ContextMenuType >, 93
FPhraseContextNode< ContextType >, 98
FPhraseEventNode, 108
FPhraseInputNode< InputType >, 113
FPhraseNode, 123
FPhraseTree, 136
IPhraseNodeBase, 173
ParsePhraseAsContext
FPhraseContextMenuNode< ContextMenuType >, 94
FPhraseContextNode< ContextType >, 99
FPhraseNode, 124
IPhraseNodeBase, 173
ParsePhraseIfRequired
FPhraseNode, 124
ParseTranscription
FPhraseTree, 137
PeekContextObject
FPhraseTreeContextManager, 143
PerformGraphAction

UAccessibilityAddNodeContextMenu, 215
PhraseInputs
FParseRecord, 83
PhraseRecord
FParseRecord, 83
PhraseTree
FOpenAccessibilityCommunicationModule, 66
UPhraseTreeUtils, 299
PhraseTreeUtils
FOpenAccessibilityCommunicationModule, 67
PinConnect
UNodeInteractionLibrary, 274
PinDisconnect
UNodeInteractionLibrary, 275
Poller
FSocketCommunicationServer, 161
poller
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 20
poller_timeout_time
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 20
PollTimeout
FSocketCommunicationServer, 161
PopContextObj
FParseRecord, 81
PopContextObject
FPhraseTreeContextManager, 144
PrevFilterString
UAccessibilityAddNodeContextMenu, 220
PreviousPositions
UAccessibilityGraphLocomotionContext, 243
PrevNumGeneratedChildren
UAccessibilityAddNodeContextMenu, 220
UAccessibilityGraphEditorContext::FTreeViewTickRequirements, 169
PrevNumItemsBeingObserved
UAccessibilityAddNodeContextMenu, 220
UAccessibilityGraphEditorContext::FTreeViewTickRequirements, 169
PrevScrollDistance
UAccessibilityAddNodeContextMenu, 220
UAccessibilityGraphEditorContext::FTreeViewTickRequirements, 169
PrevSearchText
UAccessibilityGraphEditorContext::FTreeViewTickRequirements, 170
PRIVATE_OnAudioGenerate
UAudioManager, 250
process_audio_buffer
OpenAccessibilityPy.WhisperInterface.WhisperInterface, 309
process_file_from_dir
OpenAccessibilityPy.WhisperInterface.WhisperInterface, 309
PushContextObj
FParseRecord, 82
PushContextObject
FPhraseTreeContextManager, 145
pyshutdown_handle
OpenAccessibilityPy.OpenAccessibilityPy, 186

ReachedNode
FParseResult, 85
ReceiveJSON
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 13
ReceiveMultipart
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 14
ReceiveNDArray
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 14
ReceiveNDArrayWithMeta
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 15
ReceiveString
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 15
RecieveRaw
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 16
RecordInput
FPhraseEnumInputNode< TEnum >, 105
FPhraseInputNode< InputType >, 114
FPhraseStringInputNode, 133
recv_socket
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 20
recv_socket_context
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 20
RecvAddress
FSocketCommunicationServer, 162
RecvArray
FSocketCommunicationServer, 149
RecvJson
FSocketCommunicationServer, 150
RecvMultipartWithMeta
FSocketCommunicationServer, 150
RecvSocket
FSocketCommunicationServer, 162
RecvString
FSocketCommunicationServer, 151
RecvStringMultipart
FSocketCommunicationServer, 152
RecvStringMultipartWithMeta
FSocketCommunicationServer, 152
RefreshAccessibilityWidgets
UAccessibilityAddNodeContextMenu, 215
RegisteredLibraries
UPhraseTreeUtils, 299
RegisterFunctionLibrary
UPhraseTreeUtils, 298
RegisterGameWorldAsset
FAssetAccessibilityRegistry, 29
RegisterGraphAsset

FAssetAccessibilityRegistry, 29, 30
RegisterTicker
FTranscriptionVisualizer, 166
RemoveContextObj
FParseRecord, 82
RemoveMenuDismissed
UPhraseTreeContextMenuObject, 291
RemoveNode
FGraphIndexer, 44, 45
RemovePhraseInput
FParseRecord, 82
RemoveTickDelegate
UPhraseTreeContextMenuObject, 291
RemoveValue
FIndexer< KeyType, ValueType >, 58
RemoveVisualGrid
UAccessibilityGraphLocomotionContext, 240
ReparentWindow
FTranscriptionVisualizer, 166
RequiresPhrase
FPhraseEventNode, 108, 109
FPhraseInputNode< InputType >, 114, 115
FPhraseNode, 124, 125
IPhraseNodeBase, 173
resample
OpenAccessibilityPy.Audio.AudioResampler, 10
Reset
FIndexer< KeyType, ValueType >, 59
ResetFilterText
UAccessibilityAddNodeContextMenu, 216
Result
FParseResult, 85
RevertToPreviousView
UAccessibilityGraphLocomotionContext, 240

SAccessibilityTranscriptionVis, 186
∼SAccessibilityTranscriptionVis, 187
Construct, 187
SLATE_BEGIN_ARGS, 188
Tick, 188
TranscriptionContainer, 189
TranscriptionSlots, 189
UpdateTopTranscription, 189
SaveAudioBufferToWAV
UAudioManager, 251
SaveName
FAudioManagerSettings, 34
SavePath
FAudioManagerSettings, 34
ScaleMenu
UAccessibilityAddNodeContextMenu, 216
UAccessibilityGraphEditorContext, 227
UPhraseTreeContextMenuObject, 291
SContentIndexer, 190
∼SContentIndexer, 191
Construct, 191
ConstructBottomIndexer, 191
ConstructContentContainer, 192
ConstructIndexContainer, 193
ConstructIndexText, 193
ConstructLeftIndexer, 193
ConstructRightIndexer, 194
ConstructTopIndexer, 195
GetContent, 195, 196
IndexedContent, 197
IndexerWidget, 197
SLATE_BEGIN_ARGS, 196
Tick, 196
UpdateIndex, 197
SelectAction
UAccessibilityGraphEditorContext, 228
SelectChunk
UAccessibilityGraphLocomotionContext, 241
SelectGraphAction
UAccessibilityAddNodeContextMenu, 217
SelectionAlignment
UNodeInteractionLibrary, 276
SelectionComment
UNodeInteractionLibrary, 276
SelectionMove
UNodeInteractionLibrary, 277
SelectionNodeToggle
UNodeInteractionLibrary, 278
SelectionReset
UNodeInteractionLibrary, 278
SelectionStraighten
UNodeInteractionLibrary, 278
SelectToolBarItem
UWindowInteractionLibrary, 306
SelectToolbarItem
UAccessibilityWindowToolbar, 246
send_socket_context
OpenAccessibilityPy.CommunicationServer.CommunicationServer,
20
SendAddress
FSocketCommunicationServer, 162
SendArrayBuffer
FSocketCommunicationServer, 153, 154
SendArrayMessage
FSocketCommunicationServer, 155, 156
SendArrayMessageWithMeta
FSocketCommunicationServer, 157, 158
SendJSON
OpenAccessibilityPy.CommunicationServer.CommunicationServer,
16
SendJsonBuffer
FSocketCommunicationServer, 159
SendMultipart
OpenAccessibilityPy.CommunicationServer.CommunicationServer,
17
SendMultipartWithMeta
OpenAccessibilityPy.CommunicationServer.CommunicationServer,
17
SendNDArray
OpenAccessibilityPy.CommunicationServer.CommunicationServer,
18
SendNDArrayWithMeta

OpenAccessibilityPy.CommunicationServer.CommunicationServer, 18
SendSocket
FSocketCommunicationServer, 162
SendString
OpenAccessibilityPy.CommunicationServer.CommunicationServer, 19
SendStringBuffer
FSocketCommunicationServer, 160
SerializeJSON
FSocketCommunicationServer, 161
SetChunkBounds
FGraphLocomotionChunk, 47
SetContextRootNode
UPhraseTreeContextObject, 295
SetEnumType
UParseEnumInput, 281
SetFilterText
UAccessibilityAddNodeContextMenu, 217
UAccessibilityGraphEditorContext, 229
SetMenu
UPhraseTreeContextMenuObject, 292
SetPhraseTree
UPhraseTreeUtils, 299
SetScrollDistance
UAccessibilityAddNodeContextMenu, 217
UAccessibilityGraphEditorContext, 229
SetScrollDistanceBottom
UAccessibilityAddNodeContextMenu, 218
UAccessibilityGraphEditorContext, 230
SetScrollDistanceTop
UAccessibilityAddNodeContextMenu, 218
UAccessibilityGraphEditorContext, 230
SetSharedPtr
FAccessibilityNodeFactory, 22
Settings
UAudioManager, 252
SetValue
UParseIntInput, 284
UParseStringInput, 286
SetVisColor
FGraphLocomotionChunk, 48
Shutdown
OpenAccessibilityPy.OpenAccessibilityPy, 184
ShutdownModule
FOpenAccessibilityAnalyticsModule, 61
FOpenAccessibilityCommunicationModule, 64
FOpenAccessibilityModule, 68
SIndexer, 198
∼SIndexer, 198
Construct, 199
GetIndexText, 199
IndexTextBlock, 201
SLATE_BEGIN_ARGS, 199
Tick, 199
UpdateIndex, 200
SLATE_BEGIN_ARGS
SAccessibilityTranscriptionVis, 188
SContentIndexer, 196
SIndexer, 199
SocketServer
FOpenAccessibilityCommunicationModule, 67
Source/OpenAccessibility/OpenAccessibility.Build.cs, 325
Source/OpenAccessibility/Private/AccessibilityWidgets/SAccessibilityTrans
326
Source/OpenAccessibility/Private/AccessibilityWidgets/SContentIndexer.cp
327
Source/OpenAccessibility/Private/AccessibilityWidgets/SIndexer.cpp, 329
Source/OpenAccessibility/Private/AccessibilityWrappers/AccessibilityAddN
330
Source/OpenAccessibility/Private/AccessibilityWrappers/AccessibilityGrap
334
Source/OpenAccessibility/Private/AccessibilityWrappers/AccessibilityGrap
339
Source/OpenAccessibility/Private/AccessibilityWrappers/AccessibilityWind
343
Source/OpenAccessibility/Private/AssetAccessibilityRegistry.cpp, 347
Source/OpenAccessibility/Private/GraphIndexer.cpp, 350
Source/OpenAccessibility/Private/OAccessibilityNodeFactory.cpp, 353
Source/OpenAccessibility/Private/OAEditorAccessibilityManager.cpp, 356
Source/OpenAccessibility/Private/OpenAccessibility.cpp, 356
Source/OpenAccessibility/Private/PhraseEvents/LocalizedInputLibrary.cpp
360
Source/OpenAccessibility/Private/PhraseEvents/NodeInteractionLibrary.cp
363
Source/OpenAccessibility/Private/PhraseEvents/ViewInteractionLibrary.cp
375
Source/OpenAccessibility/Private/PhraseEvents/WindowInteractionLibrary
377
Source/OpenAccessibility/Private/TranscriptionVisualizer.cpp, 378
Source/OpenAccessibility/Private/Utils/WidgetUtils.h, 380
Source/OpenAccessibility/Public/AccessibilityNodeFactory.h, 382
Source/OpenAccessibility/Public/AccessibilityWidgets/SAccessibilityTrans
384
Source/OpenAccessibility/Public/AccessibilityWidgets/SContentIndexer.h, 384
Source/OpenAccessibility/Public/AccessibilityWidgets/SIndexer.h, 385
Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityAddN
386
Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityGraph
387
Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityGraph
389
Source/OpenAccessibility/Public/AccessibilityWrappers/AccessibilityWindo
391

Source/OpenAccessibility/Public/AssetAccessibilityRegistry.h, 392
Source/OpenAccessibility/Public/GraphIndexer.h, 393
Source/OpenAccessibility/Public/Indexers/Indexer.h, 394
Source/OpenAccessibility/Public/OAccessibilityNodeFactory.h, 396
Source/OpenAccessibility/Public/OAEditorAccessibilityManager.h, 397
Source/OpenAccessibility/Public/OpenAccessibility.h, 397
Source/OpenAccessibility/Public/OpenAccessibilityLogging.h, 398
Source/OpenAccessibility/Public/PhraseEvents/LocalizedInputLibrary.h, 398
Source/OpenAccessibility/Public/PhraseEvents/NodeInteractionLibrary.h, 399
Source/OpenAccessibility/Public/PhraseEvents/Utils.h, 452
Source/OpenAccessibility/Public/PhraseEvents/ViewInteractionLibrary.h, 400
Source/OpenAccessibility/Public/PhraseEvents/WindowInteractionLibrary.h, 400
Source/OpenAccessibility/Public/TranscriptionVisualizer.h, 401
Source/OpenAccessibilityAnalytics/OpenAccessibilityAnalytics.Build.cs, 402
Source/OpenAccessibilityAnalytics/Private/OpenAccessibilityAnalytics.cpp, 402
Source/OpenAccessibilityAnalytics/Private/OpenAccessibilityAnalyticsLogging.h, 404
Source/OpenAccessibilityAnalytics/Public/OpenAccessibilityAnalytics.h, 404
Source/OpenAccessibilityCommunication/OpenAccessibilityCommunication.Build.cs, 406
Source/OpenAccessibilityCommunication/Private/AudioManager.cpp, 407
Source/OpenAccessibilityCommunication/Private/OpenAccessibilityComLogging.cpp, 408
Source/OpenAccessibilityCommunication/Private/OpenAccessibilityCommunication.cpp, 408
Source/OpenAccessibilityCommunication/Private/PhraseTree.cpp, 411
Source/OpenAccessibilityCommunication/Private/PhraseTree/Containers/ContextMenuObject.cpp, 414
Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseEnumInputNode.cpp, 415
Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseEventNode.cpp, 416
Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseInputNode.cpp, 417
Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseNode.cpp, 418
Source/OpenAccessibilityCommunication/Private/PhraseTree/PhraseStringInputNode.cpp, 421
Source/OpenAccessibilityCommunication/Private/PhraseTree/Utils.cpp, 422
Source/OpenAccessibilityCommunication/Private/PhraseTreeUtils.cpp, 423
Source/OpenAccessibilityCommunication/Private/SocketCommunicationS 423
Source/OpenAccessibilityCommunication/Private/UBAudioCapture.cpp, 429
Source/OpenAccessibilityCommunication/Public/AudioManager.h, 430
Source/OpenAccessibilityCommunication/Public/OpenAccessibilityComLo 431
Source/OpenAccessibilityCommunication/Public/OpenAccessibilityCommu 431
Source/OpenAccessibilityCommunication/Public/PhraseTree.h, 432
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/C 434
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/C 435
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/In 436
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/In 437
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/In 437
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/In 438
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/In 438
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/P 439
Source/OpenAccessibilityCommunication/Public/PhraseTree/Containers/P 442
Source/OpenAccessibilityCommunication/Public/PhraseTree/IPhraseConte 442
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseConte 442
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseConte 445
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseDirect 447
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseEnum 449
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseEvent 449
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseInputN 450
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseNode. 450
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseString 451
Source/OpenAccessibilityCommunication/Public/PhraseTree/PhraseTreeF 452
Source/OpenAccessibilityCommunication/Public/PhraseTree/Utils.h, 454
Source/OpenAccessibilityCommunication/Public/PhraseTreeUtils.h, 454
Source/OpenAccessibilityCommunication/Public/SocketCommunicationSe 455
Source/OpenAccessibilityCommunication/Public/UBAudioCapture.h,

456
StartCapturingAudio
UAudioManager, 251
StartNode
FPhraseTreeBranchBind, 141
StartupModule
FOpenAccessibilityAnalyticsModule, 62
FOpenAccessibilityCommunicationModule, 64
FOpenAccessibilityModule, 68
StartViewPosition
UAccessibilityGraphLocomotionContext, 243
StartViewZoom
UAccessibilityGraphLocomotionContext, 243
StopCapturingAudio
UAudioManager, 251
StringToNumeric
NumericParser, 176
SupportsDynamicReloading
FOpenAccessibilityAnalyticsModule, 62
FOpenAccessibilityCommunicationModule, 65
FOpenAccessibilityModule, 69

TestWhisper.ModelInfo, 175
TGraphAccessibilityNodeFactory
TGraphAccessibilityNodeFactory< T >, 202
TGraphAccessibilityNodeFactory< T >, 201
∼TGraphAccessibilityNodeFactory, 202
AccessibilityRegistry, 205
CreateNodeWidget, 203
CreatePinWidget, 204
Implementation, 205
TGraphAccessibilityNodeFactory, 202
Tick
FOpenAccessibilityCommunicationModule, 65
FPhraseTree, 139
FTranscriptionVisualizer, 167
OpenAccessibilityPy.OpenAccessibilityPy, 185
SAccessibilityTranscriptionVis, 188
SContentIndexer, 196
SIndexer, 199
UAccessibilityAddNodeContextMenu, 218
UAccessibilityGraphEditorContext, 230
UAccessibilityWindowToolbar, 246
UPhraseTreeContextMenuObject, 292
tick_handle
OpenAccessibilityPy.OpenAccessibilityPy, 186
TickDelegateHandle
FTranscriptionVisualizer, 168
TickTreeViewAccessibility
UAccessibilityGraphEditorContext, 230
ToggleContextAwareness
UAccessibilityAddNodeContextMenu, 218
TopLeft
FGraphLocomotionChunk, 49
FPanelViewPosition, 71
TranscribeWaveForm
FOpenAccessibilityCommunicationModule, 65
TranscriptionContainer
SAccessibilityTranscriptionVis, 189
TranscriptionSlots
SAccessibilityTranscriptionVis, 189
TreeView
UAccessibilityAddNodeContextMenu, 220
UAccessibilityGraphEditorContext, 233
TreeViewCanTick
UAccessibilityGraphEditorContext, 231
TreeViewRequiresTick
UAccessibilityGraphEditorContext, 231
TreeViewTickRequirements
UAccessibilityGraphEditorContext, 233

UAccessibilityAddNodeContextMenu, 206
∼UAccessibilityAddNodeContextMenu, 209
AppendFilterText, 209
AppendScrollDistance, 209
ApplyAccessibilityWidget, 210
Close, 210
ContextAwarenessCheckBox, 219
DoesItemsRequireRefresh, 211
FilterTextBox, 219
GetFilterText, 211
GetGraphActionFromIndex, 211, 212
GetGraphActionFromIndexSP, 212
GraphMenu, 219
Init, 213, 214
PerformGraphAction, 215
PrevFilterString, 220
PrevNumGeneratedChildren, 220
PrevNumItemsBeingObserved, 220
PrevScrollDistance, 220
RefreshAccessibilityWidgets, 215
ResetFilterText, 216
ScaleMenu, 216
SelectGraphAction, 217
SetFilterText, 217
SetScrollDistance, 217
SetScrollDistanceBottom, 218
SetScrollDistanceTop, 218
Tick, 218
ToggleContextAwareness, 218
TreeView, 220
UAccessibilityAddNodeContextMenu, 208
UpdateAccessibilityWidget, 219
UAccessibilityGraphEditorContext, 221
AppendFilterText, 222
AppendScrollDistance, 223
CheckBoxes, 232
Close, 223
CreateAccessibilityWrapper, 223
FilterTextBox, 233
FindGraphActionMenu, 224
FindStaticComponents, 224
FindTreeView, 225
GetFilterText, 226
GetStaticIndexOffset, 226
GetTreeViewAction, 226
GraphMenu, 233
Init, 227

ScaleMenu, 227
SelectAction, 228
SetFilterText, 229
SetScrollDistance, 229
SetScrollDistanceBottom, 230
SetScrollDistanceTop, 230
Tick, 230
TickTreeViewAccessibility, 230
TreeView, 233
TreeViewCanTick, 231
TreeViewRequiresTick, 231
TreeViewTickRequirements, 233
UAccessibilityGraphEditorContext, 222
UpdateAccessibilityWidget, 232
UAccessibilityGraphEditorContext::FTreeViewTickRequirements, 169
FTreeViewTickRequirements, 169
PrevNumGeneratedChildren, 169
PrevNumItemsBeingObserved, 169
PrevScrollDistance, 169
PrevSearchText, 170
UAccessibilityGraphLocomotionContext, 233
∼UAccessibilityGraphLocomotionContext, 235
BindFocusChangedEvent, 235
CalculateVisualChunksBounds, 235
CancelLocomotion, 236
ChangeChunkVis, 236
ChunkArray, 242
ChunkSize, 242
Close, 236
ConfirmSelection, 237
CreateVisualGrid, 237
CurrentViewPosition, 242
GenerateVisualChunks, 237
GridContainer, 243
GridParent, 243
HideNativeVisuals, 238
Init, 238, 239
LinkedEditor, 243
MoveViewport, 239
OnFocusChanged, 240
PreviousPositions, 243
RemoveVisualGrid, 240
RevertToPreviousView, 240
SelectChunk, 241
StartViewPosition, 243
StartViewZoom, 243
UAccessibilityGraphLocomotionContext, 234
UnbindFocusChangedEvent, 241
UnHideNativeVisuals, 242
UAccessibilityWindowToolbar, 244
∼UAccessibilityWindowToolbar, 245
GetActiveToolkitWidget, 245
IsActiveToolbar, 245
SelectToolbarItem, 246
Tick, 246
UAccessibilityWindowToolbar, 244
UAudioManager, 247
∼UAudioManager, 248
GetAudioCaptureNumChannels, 249
GetAudioCaptureSampleRate, 249
IsCapturingAudio, 249
OnAudioReadyForTranscription, 252
OnDefaultDeviceChanged, 250
PRIVATE_OnAudioGenerate, 250
SaveAudioBufferToWAV, 251
Settings, 252
StartCapturingAudio, 251
StopCapturingAudio, 251
UAudioManager, 248
UBAudioCapture, 252
∼UBAudioCapture, 253
OpenDefaultAudioStream, 253
UBAudioCapture, 253
ULocalizedInputLibrary, 254
∼ULocalizedInputLibrary, 255
BindBranches, 255
KeyboardInputAdd, 256
KeyboardInputConfirm, 257
KeyboardInputExit, 258
KeyboardInputRemove, 258
KeyboardInputReset, 259
ULocalizedInputLibrary, 255
UnbindFocusChangedEvent
UAccessibilityGraphLocomotionContext, 241
UnHideNativeVisuals
UAccessibilityGraphLocomotionContext, 242
UNodeInteractionLibrary, 260
∼UNodeInteractionLibrary, 261
BindBranches, 261
BlueprintCompile, 265
DeleteNode, 266
LocomotionCancel, 266
LocomotionConfirm, 267
LocomotionRevert, 267
LocomotionSelect, 267
MoveNode, 268
NodeAddMenu, 269
NodeAddPinMenu, 270
NodeAddScroll, 271
NodeAddSearchAdd, 272
NodeAddSearchRemove, 273
NodeAddSearchReset, 273
NodeAddSelect, 273
NodeIndexFocus, 274
PinConnect, 274
PinDisconnect, 275
SelectionAlignment, 276
SelectionComment, 276
SelectionMove, 277
SelectionNodeToggle, 278
SelectionReset, 278
SelectionStraighten, 278
UNodeInteractionLibrary, 261
UnregisterGameWorldAsset
FAssetAccessibilityRegistry, 30

UnregisterGraphAsset
FAssetAccessibilityRegistry, 32
UnregisterTicker
FTranscriptionVisualizer, 167
UParseEnumInput, 279
∼UParseEnumInput, 280
EnumType, 281
GetEnumType, 280
SetEnumType, 281
UParseInput, 281
∼UParseInput, 282
UParseIntInput, 282
∼UParseIntInput, 283
GetValue, 283
SetValue, 284
Value, 284
UParseStringInput, 284
∼UParseStringInput, 285
GetValue, 285
SetValue, 286
Value, 286
UpdateAccessibilityWidget
UAccessibilityAddNodeContextMenu, 219
UAccessibilityGraphEditorContext, 232
UpdateIndex
SContentIndexer, 197
SIndexer, 200
UpdateTopTranscription
SAccessibilityTranscriptionVis, 189
UpdateVisualizer
FTranscriptionVisualizer, 167
UPhraseTreeContextMenuObject, 286
∼UPhraseTreeContextMenuObject, 288
BindMenuDismissed, 288
BindTickDelegate, 289
Close, 289
GetWindow, 289
Init, 289, 290
Menu, 292
OnMenuDismissed, 290
RemoveMenuDismissed, 291
RemoveTickDelegate, 291
ScaleMenu, 291
SetMenu, 292
Tick, 292
UPhraseTreeContextMenuObject, 287, 288
Window, 292
UPhraseTreeContextObject, 293
∼UPhraseTreeContextObject, 294
bIsActive, 295
Close, 294
ContextRoot, 295
GetContextRoot, 294
GetIsActive, 294
SetContextRootNode, 295
UPhraseTreeContextObject, 294
UPhraseTreeFunctionLibrary, 296
BindBranches, 296
UPhraseTreeUtils, 297
∼UPhraseTreeUtils, 297
PhraseTree, 299
RegisteredLibraries, 299
RegisterFunctionLibrary, 298
SetPhraseTree, 299
UPhraseTreeUtils, 297
UViewInteractionLibrary, 300
∼UViewInteractionLibrary, 300
BindBranches, 301
IndexFocus, 302
MoveViewport, 302
UViewInteractionLibrary, 300
ZoomViewport, 303
UWindowInteractionLibrary, 304
∼UWindowInteractionLibrary, 305
BindBranches, 305
CloseActiveWindow, 306
SelectToolBarItem, 306
UWindowInteractionLibrary, 305
WindowToolBar, 307

Value
UParseIntInput, 284
UParseStringInput, 286
VisContent
FTranscriptionVisualizer, 168
VisWindow
FTranscriptionVisualizer, 168

WARNING
OpenAccessibilityPy.Logging.LogLevel, 175
whisper_interface
OpenAccessibilityPy.OpenAccessibilityPy, 186
whisper_model
OpenAccessibilityPy.WhisperInterface.WhisperInterface, 310
Window
UPhraseTreeContextMenuObject, 292
WindowToolBar
UWindowInteractionLibrary, 307
worker_pool
OpenAccessibilityPy.OpenAccessibilityPy, 186
WrapNodeWidget
FAccessibilityNodeFactory, 23
WrapPinWidget
FAccessibilityNodeFactory, 24

ZoomViewport
UViewInteractionLibrary, 303