

Tang

0.1

Generated by Doxygen 1.9.1

1 Tang: A Template Language	1
1.1 Quick Description	1
1.2 Features	1
1.3 License	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	9
4.1 File List	9
5 Class Documentation	15
5.1 Tang::AstNode Class Reference	15
5.1.1 Detailed Description	18
5.1.2 Member Enumeration Documentation	18
5.1.2.1 PreprocessState	18
5.1.3 Constructor & Destructor Documentation	18
5.1.3.1 AstNode()	18
5.1.4 Member Function Documentation	19
5.1.4.1 compile()	19
5.1.4.2 compilePreprocess()	19
5.1.4.3 dump()	20
5.2 Tang::AstNodeArray Class Reference	21
5.2.1 Detailed Description	23
5.2.2 Member Enumeration Documentation	23
5.2.2.1 PreprocessState	23
5.2.3 Constructor & Destructor Documentation	23
5.2.3.1 AstNodeArray()	23
5.2.4 Member Function Documentation	24
5.2.4.1 compile()	24
5.2.4.2 compilePreprocess()	24
5.2.4.3 dump()	25
5.3 Tang::AstNodeAssign Class Reference	25
5.3.1 Detailed Description	27
5.3.2 Member Enumeration Documentation	27
5.3.2.1 PreprocessState	27
5.3.3 Constructor & Destructor Documentation	27
5.3.3.1 AstNodeAssign()	27
5.3.4 Member Function Documentation	28
5.3.4.1 compile()	28
5.3.4.2 compilePreprocess()	29

5.3.4.3 <code>dump()</code>	29
5.4 <code>Tang::AstNodeBinary</code> Class Reference	29
5.4.1 Detailed Description	32
5.4.2 Member Enumeration Documentation	32
5.4.2.1 <code>Operation</code>	32
5.4.2.2 <code>PreprocessState</code>	33
5.4.3 Constructor & Destructor Documentation	33
5.4.3.1 <code>AstNodeBinary()</code>	33
5.4.4 Member Function Documentation	33
5.4.4.1 <code>compile()</code>	33
5.4.4.2 <code>compilePreprocess()</code>	34
5.4.4.3 <code>dump()</code>	34
5.5 <code>Tang::AstNodeBlock</code> Class Reference	35
5.5.1 Detailed Description	37
5.5.2 Member Enumeration Documentation	37
5.5.2.1 <code>PreprocessState</code>	37
5.5.3 Constructor & Destructor Documentation	37
5.5.3.1 <code>AstNodeBlock()</code>	37
5.5.4 Member Function Documentation	38
5.5.4.1 <code>compile()</code>	38
5.5.4.2 <code>compilePreprocess()</code>	38
5.5.4.3 <code>dump()</code>	39
5.6 <code>Tang::AstNodeBoolean</code> Class Reference	39
5.6.1 Detailed Description	41
5.6.2 Member Enumeration Documentation	41
5.6.2.1 <code>PreprocessState</code>	41
5.6.3 Constructor & Destructor Documentation	41
5.6.3.1 <code>AstNodeBoolean()</code>	41
5.6.4 Member Function Documentation	42
5.6.4.1 <code>compile()</code>	42
5.6.4.2 <code>compilePreprocess()</code>	42
5.6.4.3 <code>dump()</code>	43
5.7 <code>Tang::AstNodeBreak</code> Class Reference	43
5.7.1 Detailed Description	45
5.7.2 Member Enumeration Documentation	45
5.7.2.1 <code>PreprocessState</code>	45
5.7.3 Constructor & Destructor Documentation	45
5.7.3.1 <code>AstNodeBreak()</code>	45
5.7.4 Member Function Documentation	45
5.7.4.1 <code>compile()</code>	46
5.7.4.2 <code>compilePreprocess()</code>	46
5.7.4.3 <code>dump()</code>	47

5.8 Tang::AstNodeCast Class Reference	47
5.8.1 Detailed Description	49
5.8.2 Member Enumeration Documentation	49
5.8.2.1 PreprocessState	49
5.8.2.2 Type	49
5.8.3 Constructor & Destructor Documentation	50
5.8.3.1 AstNodeCast()	50
5.8.4 Member Function Documentation	50
5.8.4.1 compile()	50
5.8.4.2 compilePreprocess()	51
5.8.4.3 dump()	51
5.9 Tang::AstNodeContinue Class Reference	52
5.9.1 Detailed Description	54
5.9.2 Member Enumeration Documentation	54
5.9.2.1 PreprocessState	54
5.9.3 Constructor & Destructor Documentation	54
5.9.3.1 AstNodeContinue()	54
5.9.4 Member Function Documentation	54
5.9.4.1 compile()	55
5.9.4.2 compilePreprocess()	55
5.9.4.3 dump()	56
5.10 Tang::AstNodeDoWhile Class Reference	56
5.10.1 Detailed Description	58
5.10.2 Member Enumeration Documentation	58
5.10.2.1 PreprocessState	58
5.10.3 Constructor & Destructor Documentation	58
5.10.3.1 AstNodeDoWhile()	58
5.10.4 Member Function Documentation	59
5.10.4.1 compile()	59
5.10.4.2 compilePreprocess()	59
5.10.4.3 dump()	60
5.11 Tang::AstNodeFloat Class Reference	60
5.11.1 Detailed Description	62
5.11.2 Member Enumeration Documentation	62
5.11.2.1 PreprocessState	62
5.11.3 Constructor & Destructor Documentation	62
5.11.3.1 AstNodeFloat()	62
5.11.4 Member Function Documentation	63
5.11.4.1 compile()	63
5.11.4.2 compilePreprocess()	63
5.11.4.3 dump()	64
5.12 Tang::AstNodeFor Class Reference	64

5.12.1 Detailed Description	66
5.12.2 Member Enumeration Documentation	66
5.12.2.1 PreprocessState	66
5.12.3 Constructor & Destructor Documentation	67
5.12.3.1 AstNodeFor()	67
5.12.4 Member Function Documentation	67
5.12.4.1 compile()	67
5.12.4.2 compilePreprocess()	68
5.12.4.3 dump()	68
5.13 Tang::AstNodeFunctionCall Class Reference	69
5.13.1 Detailed Description	70
5.13.2 Member Enumeration Documentation	70
5.13.2.1 PreprocessState	70
5.13.3 Constructor & Destructor Documentation	71
5.13.3.1 AstNodeFunctionCall()	71
5.13.4 Member Function Documentation	71
5.13.4.1 compile()	71
5.13.4.2 compilePreprocess()	72
5.13.4.3 dump()	72
5.14 Tang::AstNodeFunctionDeclaration Class Reference	72
5.14.1 Detailed Description	74
5.14.2 Member Enumeration Documentation	74
5.14.2.1 PreprocessState	74
5.14.3 Constructor & Destructor Documentation	74
5.14.3.1 AstNodeFunctionDeclaration()	74
5.14.4 Member Function Documentation	75
5.14.4.1 compile()	75
5.14.4.2 compilePreprocess()	75
5.14.4.3 dump()	76
5.15 Tang::AstNodelIdentifier Class Reference	76
5.15.1 Detailed Description	78
5.15.2 Member Enumeration Documentation	78
5.15.2.1 PreprocessState	78
5.15.3 Constructor & Destructor Documentation	79
5.15.3.1 AstNodelIdentifier()	79
5.15.4 Member Function Documentation	79
5.15.4.1 compile()	79
5.15.4.2 compilePreprocess()	80
5.15.4.3 dump()	80
5.16 Tang::AstNodelElse Class Reference	81
5.16.1 Detailed Description	83
5.16.2 Member Enumeration Documentation	83

5.16.2.1 PreprocessState	83
5.16.3 Constructor & Destructor Documentation	84
5.16.3.1 AstNodeIfElse() [1/2]	84
5.16.3.2 AstNodeIfElse() [2/2]	84
5.16.4 Member Function Documentation	84
5.16.4.1 compile()	84
5.16.4.2 compilePreprocess()	85
5.16.4.3 dump()	85
5.17 Tang::AstNodeIndex Class Reference	86
5.17.1 Detailed Description	88
5.17.2 Member Enumeration Documentation	88
5.17.2.1 PreprocessState	88
5.17.3 Constructor & Destructor Documentation	88
5.17.3.1 AstNodeIndex()	89
5.17.4 Member Function Documentation	89
5.17.4.1 compile()	89
5.17.4.2 compilePreprocess()	90
5.17.4.3 dump()	90
5.17.4.4 getCollection()	90
5.17.4.5 getIndex()	91
5.18 Tang::AstNodeInteger Class Reference	91
5.18.1 Detailed Description	93
5.18.2 Member Enumeration Documentation	93
5.18.2.1 PreprocessState	93
5.18.3 Constructor & Destructor Documentation	93
5.18.3.1 AstNodeInteger()	93
5.18.4 Member Function Documentation	94
5.18.4.1 compile()	94
5.18.4.2 compilePreprocess()	94
5.18.4.3 dump()	95
5.19 Tang::AstNodeLibrary Class Reference	95
5.19.1 Detailed Description	97
5.19.2 Member Enumeration Documentation	97
5.19.2.1 PreprocessState	97
5.19.3 Constructor & Destructor Documentation	97
5.19.3.1 AstNodeLibrary()	97
5.19.4 Member Function Documentation	98
5.19.4.1 compile()	98
5.19.4.2 compilePreprocess()	98
5.19.4.3 dump()	99
5.20 Tang::AstNodeMap Class Reference	99
5.20.1 Detailed Description	101

5.20.2 Member Enumeration Documentation	101
5.20.2.1 PreprocessState	101
5.20.3 Constructor & Destructor Documentation	101
5.20.3.1 AstNodeMap()	101
5.20.4 Member Function Documentation	101
5.20.4.1 compile()	102
5.20.4.2 compilePreprocess()	102
5.20.4.3 dump()	103
5.21 Tang::AstNodePeriod Class Reference	103
5.21.1 Detailed Description	105
5.21.2 Member Enumeration Documentation	105
5.21.2.1 PreprocessState	105
5.21.3 Constructor & Destructor Documentation	105
5.21.3.1 AstNodePeriod()	105
5.21.4 Member Function Documentation	106
5.21.4.1 compile()	106
5.21.4.2 compilePreprocess()	106
5.21.4.3 dump()	107
5.22 Tang::AstNodePrint Class Reference	107
5.22.1 Detailed Description	109
5.22.2 Member Enumeration Documentation	109
5.22.2.1 PreprocessState	109
5.22.2.2 Type	110
5.22.3 Constructor & Destructor Documentation	110
5.22.3.1 AstNodePrint()	110
5.22.4 Member Function Documentation	110
5.22.4.1 compile()	110
5.22.4.2 compilePreprocess()	111
5.22.4.3 dump()	111
5.23 Tang::AstNodeRangedFor Class Reference	112
5.23.1 Detailed Description	113
5.23.2 Member Enumeration Documentation	113
5.23.2.1 PreprocessState	113
5.23.3 Constructor & Destructor Documentation	114
5.23.3.1 AstNodeRangedFor()	114
5.23.4 Member Function Documentation	114
5.23.4.1 compile()	114
5.23.4.2 compilePreprocess()	115
5.23.4.3 dump()	116
5.24 Tang::AstNodeReturn Class Reference	116
5.24.1 Detailed Description	118
5.24.2 Member Enumeration Documentation	118

5.24.2.1 PreprocessState	118
5.24.3 Constructor & Destructor Documentation	118
5.24.3.1 AstNodeReturn()	118
5.24.4 Member Function Documentation	119
5.24.4.1 compile()	119
5.24.4.2 compilePreprocess()	119
5.24.4.3 dump()	120
5.25 Tang::AstNodeSlice Class Reference	120
5.25.1 Detailed Description	122
5.25.2 Member Enumeration Documentation	122
5.25.2.1 PreprocessState	122
5.25.3 Constructor & Destructor Documentation	123
5.25.3.1 AstNodeSlice()	123
5.25.4 Member Function Documentation	123
5.25.4.1 compile()	123
5.25.4.2 compilePreprocess()	124
5.25.4.3 dump()	124
5.26 Tang::AstNodeString Class Reference	125
5.26.1 Detailed Description	127
5.26.2 Member Enumeration Documentation	127
5.26.2.1 PreprocessState	127
5.26.3 Constructor & Destructor Documentation	127
5.26.3.1 AstNodeString() [1/2]	128
5.26.3.2 AstNodeString() [2/2]	128
5.26.4 Member Function Documentation	128
5.26.4.1 compile()	128
5.26.4.2 compileLiteral()	129
5.26.4.3 compilePreprocess()	129
5.26.4.4 dump()	130
5.27 Tang::AstNodeTernary Class Reference	130
5.27.1 Detailed Description	133
5.27.2 Member Enumeration Documentation	133
5.27.2.1 PreprocessState	133
5.27.3 Constructor & Destructor Documentation	133
5.27.3.1 AstNodeTernary()	134
5.27.4 Member Function Documentation	134
5.27.4.1 compile()	134
5.27.4.2 compilePreprocess()	135
5.27.4.3 dump()	135
5.28 Tang::AstNodeUnary Class Reference	135
5.28.1 Detailed Description	137
5.28.2 Member Enumeration Documentation	137

5.28.2.1 Operator	137
5.28.2.2 PreprocessState	138
5.28.3 Constructor & Destructor Documentation	138
5.28.3.1 AstNodeUnary()	138
5.28.4 Member Function Documentation	138
5.28.4.1 compile()	138
5.28.4.2 compilePreprocess()	140
5.28.4.3 dump()	140
5.29 Tang::AstNodeUse Class Reference	141
5.29.1 Detailed Description	143
5.29.2 Member Enumeration Documentation	143
5.29.2.1 PreprocessState	143
5.29.3 Constructor & Destructor Documentation	143
5.29.3.1 AstNodeUse()	143
5.29.4 Member Function Documentation	144
5.29.4.1 compile()	144
5.29.4.2 compilePreprocess()	144
5.29.4.3 dump()	145
5.30 Tang::AstNodeWhile Class Reference	145
5.30.1 Detailed Description	148
5.30.2 Member Enumeration Documentation	148
5.30.2.1 PreprocessState	148
5.30.3 Constructor & Destructor Documentation	148
5.30.3.1 AstNodeWhile()	148
5.30.4 Member Function Documentation	149
5.30.4.1 compile()	149
5.30.4.2 compilePreprocess()	150
5.30.4.3 dump()	150
5.31 Tang::ComputedExpression Class Reference	150
5.31.1 Detailed Description	153
5.31.2 Member Function Documentation	153
5.31.2.1 __add()	153
5.31.2.2 __asCode()	153
5.31.2.3 __assign_index()	153
5.31.2.4 __boolean()	154
5.31.2.5 __divide()	154
5.31.2.6 __equal()	155
5.31.2.7 __float()	155
5.31.2.8 __getIterator()	155
5.31.2.9 __index()	156
5.31.2.10 __integer()	156
5.31.2.11 __iteratorNext()	156

5.31.2.12 <code>__lessThan()</code>	157
5.31.2.13 <code>__modulo()</code>	157
5.31.2.14 <code>__multiply()</code>	157
5.31.2.15 <code>__negative()</code>	158
5.31.2.16 <code>__not()</code>	158
5.31.2.17 <code>__period()</code>	158
5.31.2.18 <code>__slice()</code>	159
5.31.2.19 <code>__string()</code>	159
5.31.2.20 <code>__subtract()</code>	159
5.31.2.21 <code>dump()</code>	160
5.31.2.22 <code>is_equal()</code> [1/6]	160
5.31.2.23 <code>is_equal()</code> [2/6]	161
5.31.2.24 <code>is_equal()</code> [3/6]	161
5.31.2.25 <code>is_equal()</code> [4/6]	161
5.31.2.26 <code>is_equal()</code> [5/6]	162
5.31.2.27 <code>is_equal()</code> [6/6]	162
5.31.2.28 <code>isCopyNeeded()</code>	162
5.31.2.29 <code>makeCopy()</code>	163
5.32 <code>Tang::ComputedExpressionArray</code> Class Reference	163
5.32.1 Detailed Description	166
5.32.2 Constructor & Destructor Documentation	166
5.32.2.1 <code>ComputedExpressionArray()</code>	166
5.32.3 Member Function Documentation	166
5.32.3.1 <code>__add()</code>	166
5.32.3.2 <code>__asCode()</code>	167
5.32.3.3 <code>__assign_index()</code>	167
5.32.3.4 <code>__boolean()</code>	168
5.32.3.5 <code>__divide()</code>	168
5.32.3.6 <code>__equal()</code>	168
5.32.3.7 <code>__float()</code>	169
5.32.3.8 <code>__getIterator()</code>	169
5.32.3.9 <code>__index()</code>	169
5.32.3.10 <code>__integer()</code>	170
5.32.3.11 <code>__iteratorNext()</code>	170
5.32.3.12 <code>__lessThan()</code>	171
5.32.3.13 <code>__modulo()</code>	171
5.32.3.14 <code>__multiply()</code>	171
5.32.3.15 <code>__negative()</code>	172
5.32.3.16 <code>__not()</code>	172
5.32.3.17 <code>__period()</code>	172
5.32.3.18 <code>__slice()</code>	173
5.32.3.19 <code>__string()</code>	174

5.32.3.20 __subtract()	174
5.32.3.21 append()	174
5.32.3.22 dump()	175
5.32.3.23 getContents()	175
5.32.3.24 getMethods()	175
5.32.3.25 is_equal() [1/6]	176
5.32.3.26 is_equal() [2/6]	176
5.32.3.27 is_equal() [3/6]	176
5.32.3.28 is_equal() [4/6]	177
5.32.3.29 is_equal() [5/6]	177
5.32.3.30 is_equal() [6/6]	177
5.32.3.31 isCopyNeeded()	178
5.32.3.32 makeCopy()	178
5.33 Tang::ComputedExpressionBoolean Class Reference	179
5.33.1 Detailed Description	181
5.33.2 Constructor & Destructor Documentation	181
5.33.2.1 ComputedExpressionBoolean()	181
5.33.3 Member Function Documentation	181
5.33.3.1 __add()	181
5.33.3.2 __asCode()	182
5.33.3.3 __assign_index()	182
5.33.3.4 __boolean()	182
5.33.3.5 __divide()	182
5.33.3.6 __equal()	183
5.33.3.7 __float()	183
5.33.3.8 __getIterator()	184
5.33.3.9 __index()	184
5.33.3.10 __integer()	184
5.33.3.11 __iteratorNext()	184
5.33.3.12 __lessThan()	185
5.33.3.13 __modulo()	185
5.33.3.14 __multiply()	186
5.33.3.15 __negative()	186
5.33.3.16 __not()	186
5.33.3.17 __period()	186
5.33.3.18 __slice()	187
5.33.3.19 __string()	187
5.33.3.20 __subtract()	188
5.33.3.21 dump()	188
5.33.3.22 is_equal() [1/6]	188
5.33.3.23 is_equal() [2/6]	189
5.33.3.24 is_equal() [3/6]	189

5.33.3.25 <code>is_equal()</code> [4/6]	189
5.33.3.26 <code>is_equal()</code> [5/6]	190
5.33.3.27 <code>is_equal()</code> [6/6]	190
5.33.3.28 <code>isCopyNeeded()</code>	190
5.33.3.29 <code>makeCopy()</code>	191
5.34 <code>Tang::ComputedExpressionCompiledFunction</code> Class Reference	191
5.34.1 Detailed Description	193
5.34.2 Constructor & Destructor Documentation	193
5.34.2.1 <code>ComputedExpressionCompiledFunction()</code>	194
5.34.3 Member Function Documentation	194
5.34.3.1 <code>__add()</code>	194
5.34.3.2 <code>__asCode()</code>	194
5.34.3.3 <code>__assign_index()</code>	195
5.34.3.4 <code>__boolean()</code>	195
5.34.3.5 <code>__divide()</code>	195
5.34.3.6 <code>__equal()</code>	196
5.34.3.7 <code>__float()</code>	196
5.34.3.8 <code>__getIterator()</code>	197
5.34.3.9 <code>__index()</code>	197
5.34.3.10 <code>__integer()</code>	197
5.34.3.11 <code>__iteratorNext()</code>	197
5.34.3.12 <code>__lessThan()</code>	198
5.34.3.13 <code>__modulo()</code>	198
5.34.3.14 <code>__multiply()</code>	199
5.34.3.15 <code>__negative()</code>	199
5.34.3.16 <code>__not()</code>	199
5.34.3.17 <code>__period()</code>	199
5.34.3.18 <code>__slice()</code>	200
5.34.3.19 <code>__string()</code>	200
5.34.3.20 <code>__subtract()</code>	201
5.34.3.21 <code>dump()</code>	201
5.34.3.22 <code>is_equal()</code> [1/6]	201
5.34.3.23 <code>is_equal()</code> [2/6]	202
5.34.3.24 <code>is_equal()</code> [3/6]	202
5.34.3.25 <code>is_equal()</code> [4/6]	202
5.34.3.26 <code>is_equal()</code> [5/6]	203
5.34.3.27 <code>is_equal()</code> [6/6]	203
5.34.3.28 <code>isCopyNeeded()</code>	203
5.34.3.29 <code>makeCopy()</code>	204
5.35 <code>Tang::ComputedExpressionError</code> Class Reference	204
5.35.1 Detailed Description	207
5.35.2 Constructor & Destructor Documentation	207

5.35.2.1 <code>ComputedExpressionError()</code>	207
5.35.3 Member Function Documentation	207
5.35.3.1 <code>__add()</code>	207
5.35.3.2 <code>__asCode()</code>	208
5.35.3.3 <code>__assign_index()</code>	208
5.35.3.4 <code>__boolean()</code>	208
5.35.3.5 <code>__divide()</code>	208
5.35.3.6 <code>__equal()</code>	209
5.35.3.7 <code>__float()</code>	209
5.35.3.8 <code>__getIterator()</code>	209
5.35.3.9 <code>__index()</code>	210
5.35.3.10 <code>__integer()</code>	210
5.35.3.11 <code>__iteratorNext()</code>	210
5.35.3.12 <code>__lessThan()</code>	211
5.35.3.13 <code>__modulo()</code>	211
5.35.3.14 <code>__multiply()</code>	212
5.35.3.15 <code>__negative()</code>	212
5.35.3.16 <code>__not()</code>	212
5.35.3.17 <code>__period()</code>	212
5.35.3.18 <code>__slice()</code>	213
5.35.3.19 <code>__string()</code>	213
5.35.3.20 <code>__subtract()</code>	214
5.35.3.21 <code>dump()</code>	214
5.35.3.22 <code>is_equal()</code> [1/6]	214
5.35.3.23 <code>is_equal()</code> [2/6]	215
5.35.3.24 <code>is_equal()</code> [3/6]	215
5.35.3.25 <code>is_equal()</code> [4/6]	215
5.35.3.26 <code>is_equal()</code> [5/6]	216
5.35.3.27 <code>is_equal()</code> [6/6]	216
5.35.3.28 <code>isCopyNeeded()</code>	216
5.35.3.29 <code>makeCopy()</code>	217
5.36 <code>Tang::ComputedExpressionFloat</code> Class Reference	217
5.36.1 Detailed Description	219
5.36.2 Constructor & Destructor Documentation	219
5.36.2.1 <code>ComputedExpressionFloat()</code>	219
5.36.3 Member Function Documentation	220
5.36.3.1 <code>__add()</code>	220
5.36.3.2 <code>__asCode()</code>	220
5.36.3.3 <code>__assign_index()</code>	221
5.36.3.4 <code>__boolean()</code>	221
5.36.3.5 <code>__divide()</code>	221
5.36.3.6 <code>__equal()</code>	222

5.36.3.7 <code>__float()</code>	222
5.36.3.8 <code>__getIterator()</code>	223
5.36.3.9 <code>__index()</code>	223
5.36.3.10 <code>__integer()</code>	223
5.36.3.11 <code>__iteratorNext()</code>	224
5.36.3.12 <code>__lessThan()</code>	224
5.36.3.13 <code>__modulo()</code>	224
5.36.3.14 <code>__multiply()</code>	225
5.36.3.15 <code>__negative()</code>	225
5.36.3.16 <code>__not()</code>	226
5.36.3.17 <code>__period()</code>	226
5.36.3.18 <code>__slice()</code>	226
5.36.3.19 <code>__string()</code>	227
5.36.3.20 <code>__subtract()</code>	227
5.36.3.21 <code>dump()</code>	228
5.36.3.22 <code>getValue()</code>	228
5.36.3.23 <code>is_equal()</code> [1/6]	228
5.36.3.24 <code>is_equal()</code> [2/6]	229
5.36.3.25 <code>is_equal()</code> [3/6]	229
5.36.3.26 <code>is_equal()</code> [4/6]	229
5.36.3.27 <code>is_equal()</code> [5/6]	230
5.36.3.28 <code>is_equal()</code> [6/6]	230
5.36.3.29 <code>isCopyNeeded()</code>	231
5.36.3.30 <code>makeCopy()</code>	231
5.37 Tang::ComputedExpressionInteger Class Reference	231
5.37.1 Detailed Description	233
5.37.2 Constructor & Destructor Documentation	233
5.37.2.1 ComputedExpressionInteger()	233
5.37.3 Member Function Documentation	234
5.37.3.1 <code>__add()</code>	234
5.37.3.2 <code>__asCode()</code>	234
5.37.3.3 <code>__assign_index()</code>	235
5.37.3.4 <code>__boolean()</code>	235
5.37.3.5 <code>__divide()</code>	235
5.37.3.6 <code>__equal()</code>	236
5.37.3.7 <code>__float()</code>	236
5.37.3.8 <code>__getIterator()</code>	237
5.37.3.9 <code>__index()</code>	237
5.37.3.10 <code>__integer()</code>	237
5.37.3.11 <code>__iteratorNext()</code>	238
5.37.3.12 <code>__lessThan()</code>	238
5.37.3.13 <code>__modulo()</code>	238

5.37.3.14 <code>__multiply()</code>	239
5.37.3.15 <code>__negative()</code>	240
5.37.3.16 <code>__not()</code>	240
5.37.3.17 <code>__period()</code>	240
5.37.3.18 <code>__slice()</code>	241
5.37.3.19 <code>__string()</code>	241
5.37.3.20 <code>__subtract()</code>	241
5.37.3.21 <code>dump()</code>	242
5.37.3.22 <code>getValue()</code>	242
5.37.3.23 <code>is_equal()</code> [1/6]	242
5.37.3.24 <code>is_equal()</code> [2/6]	243
5.37.3.25 <code>is_equal()</code> [3/6]	243
5.37.3.26 <code>is_equal()</code> [4/6]	244
5.37.3.27 <code>is_equal()</code> [5/6]	244
5.37.3.28 <code>is_equal()</code> [6/6]	244
5.37.3.29 <code>isCopyNeeded()</code>	245
5.37.3.30 <code>makeCopy()</code>	245
5.38 Tang::ComputedExpressionIterator Class Reference	246
5.38.1 Detailed Description	248
5.38.2 Constructor & Destructor Documentation	248
5.38.2.1 <code>ComputedExpressionIterator()</code>	248
5.38.3 Member Function Documentation	248
5.38.3.1 <code>__add()</code>	248
5.38.3.2 <code>__asCode()</code>	249
5.38.3.3 <code>__assign_index()</code>	249
5.38.3.4 <code>__boolean()</code>	250
5.38.3.5 <code>__divide()</code>	250
5.38.3.6 <code>__equal()</code>	250
5.38.3.7 <code>__float()</code>	251
5.38.3.8 <code>__getIterator()</code>	251
5.38.3.9 <code>__index()</code>	251
5.38.3.10 <code>__integer()</code>	252
5.38.3.11 <code>__iteratorNext()</code>	252
5.38.3.12 <code>__lessThan()</code>	253
5.38.3.13 <code>__modulo()</code>	253
5.38.3.14 <code>__multiply()</code>	253
5.38.3.15 <code>__negative()</code>	254
5.38.3.16 <code>__not()</code>	254
5.38.3.17 <code>__period()</code>	254
5.38.3.18 <code>__slice()</code>	255
5.38.3.19 <code>__string()</code>	255
5.38.3.20 <code>__subtract()</code>	255

5.38.3.21 <code>dump()</code>	256
5.38.3.22 <code>is_equal()</code> [1/6]	256
5.38.3.23 <code>is_equal()</code> [2/6]	257
5.38.3.24 <code>is_equal()</code> [3/6]	258
5.38.3.25 <code>is_equal()</code> [4/6]	258
5.38.3.26 <code>is_equal()</code> [5/6]	259
5.38.3.27 <code>is_equal()</code> [6/6]	259
5.38.3.28 <code>isCopyNeeded()</code>	259
5.38.3.29 <code>makeCopy()</code>	260
5.39 <code>Tang::ComputedExpressionIteratorEnd</code> Class Reference	260
5.39.1 Detailed Description	262
5.39.2 Member Function Documentation	262
5.39.2.1 <code>__add()</code>	262
5.39.2.2 <code>__asCode()</code>	263
5.39.2.3 <code>__assign_index()</code>	263
5.39.2.4 <code>__boolean()</code>	264
5.39.2.5 <code>__divide()</code>	264
5.39.2.6 <code>__equal()</code>	264
5.39.2.7 <code>__float()</code>	265
5.39.2.8 <code>__getIterator()</code>	265
5.39.2.9 <code>__index()</code>	265
5.39.2.10 <code>__integer()</code>	266
5.39.2.11 <code>__iteratorNext()</code>	266
5.39.2.12 <code>__lessThan()</code>	266
5.39.2.13 <code>__modulo()</code>	267
5.39.2.14 <code>__multiply()</code>	267
5.39.2.15 <code>__negative()</code>	268
5.39.2.16 <code>__not()</code>	268
5.39.2.17 <code>__period()</code>	268
5.39.2.18 <code>__slice()</code>	269
5.39.2.19 <code>__string()</code>	269
5.39.2.20 <code>__subtract()</code>	269
5.39.2.21 <code>dump()</code>	270
5.39.2.22 <code>is_equal()</code> [1/6]	270
5.39.2.23 <code>is_equal()</code> [2/6]	271
5.39.2.24 <code>is_equal()</code> [3/6]	272
5.39.2.25 <code>is_equal()</code> [4/6]	272
5.39.2.26 <code>is_equal()</code> [5/6]	273
5.39.2.27 <code>is_equal()</code> [6/6]	273
5.39.2.28 <code>isCopyNeeded()</code>	273
5.39.2.29 <code>makeCopy()</code>	274
5.40 <code>Tang::ComputedExpressionLibrary</code> Class Reference	274

5.40.1 Detailed Description	276
5.40.2 Member Function Documentation	276
5.40.2.1 <code>__add()</code>	276
5.40.2.2 <code>__asCode()</code>	277
5.40.2.3 <code>__assign_index()</code>	277
5.40.2.4 <code>__boolean()</code>	278
5.40.2.5 <code>__divide()</code>	278
5.40.2.6 <code>__equal()</code>	278
5.40.2.7 <code>__float()</code>	279
5.40.2.8 <code>__getIterator()</code>	279
5.40.2.9 <code>__index()</code>	279
5.40.2.10 <code>__integer()</code>	280
5.40.2.11 <code>__iteratorNext()</code>	280
5.40.2.12 <code>__lessThan()</code>	280
5.40.2.13 <code>__modulo()</code>	281
5.40.2.14 <code>__multiply()</code>	281
5.40.2.15 <code>__negative()</code>	282
5.40.2.16 <code>__not()</code>	282
5.40.2.17 <code>__period()</code>	282
5.40.2.18 <code>__slice()</code>	283
5.40.2.19 <code>__string()</code>	283
5.40.2.20 <code>__subtract()</code>	283
5.40.2.21 <code>dump()</code>	284
5.40.2.22 <code>is_equal()</code> [1/6]	284
5.40.2.23 <code>is_equal()</code> [2/6]	285
5.40.2.24 <code>is_equal()</code> [3/6]	286
5.40.2.25 <code>is_equal()</code> [4/6]	286
5.40.2.26 <code>is_equal()</code> [5/6]	287
5.40.2.27 <code>is_equal()</code> [6/6]	287
5.40.2.28 <code>isCopyNeeded()</code>	287
5.40.2.29 <code>makeCopy()</code>	288
5.41 <code>Tang::ComputedExpressionLibraryMath</code> Class Reference	288
5.41.1 Detailed Description	290
5.41.2 Member Function Documentation	290
5.41.2.1 <code>__add()</code>	291
5.41.2.2 <code>__asCode()</code>	291
5.41.2.3 <code>__assign_index()</code>	291
5.41.2.4 <code>__boolean()</code>	292
5.41.2.5 <code>__divide()</code>	292
5.41.2.6 <code>__equal()</code>	292
5.41.2.7 <code>__float()</code>	293
5.41.2.8 <code>__getIterator()</code>	293

5.41.2.9 __index()	293
5.41.2.10 __integer()	294
5.41.2.11 __iteratorNext()	294
5.41.2.12 __lessThan()	294
5.41.2.13 __modulo()	295
5.41.2.14 __multiply()	295
5.41.2.15 __negative()	296
5.41.2.16 __not()	296
5.41.2.17 __period()	296
5.41.2.18 __slice()	297
5.41.2.19 __string()	297
5.41.2.20 __subtract()	297
5.41.2.21 dump()	298
5.41.2.22 getLibraryAttributes()	298
5.41.2.23 is_equal() [1/6]	298
5.41.2.24 is_equal() [2/6]	299
5.41.2.25 is_equal() [3/6]	299
5.41.2.26 is_equal() [4/6]	299
5.41.2.27 is_equal() [5/6]	300
5.41.2.28 is_equal() [6/6]	300
5.41.2.29 isCopyNeeded()	301
5.41.2.30 makeCopy()	301
5.42 Tang::ComputedExpressionMap Class Reference	301
5.42.1 Detailed Description	303
5.42.2 Constructor & Destructor Documentation	304
5.42.2.1 ComputedExpressionMap()	304
5.42.3 Member Function Documentation	304
5.42.3.1 __add()	304
5.42.3.2 __asCode()	304
5.42.3.3 __assign_index()	305
5.42.3.4 __boolean()	305
5.42.3.5 __divide()	305
5.42.3.6 __equal()	306
5.42.3.7 __float()	306
5.42.3.8 __getIterator()	306
5.42.3.9 __index()	307
5.42.3.10 __integer()	307
5.42.3.11 __iteratorNext()	308
5.42.3.12 __lessThan()	308
5.42.3.13 __modulo()	308
5.42.3.14 __multiply()	309
5.42.3.15 __negative()	309

5.42.3.16 <code>__not()</code>	309
5.42.3.17 <code>__period()</code>	309
5.42.3.18 <code>__slice()</code>	310
5.42.3.19 <code>__string()</code>	310
5.42.3.20 <code>__subtract()</code>	311
5.42.3.21 <code>dump()</code>	311
5.42.3.22 <code>is_equal()</code> [1/6]	312
5.42.3.23 <code>is_equal()</code> [2/6]	312
5.42.3.24 <code>is_equal()</code> [3/6]	312
5.42.3.25 <code>is_equal()</code> [4/6]	313
5.42.3.26 <code>is_equal()</code> [5/6]	313
5.42.3.27 <code>is_equal()</code> [6/6]	313
5.42.3.28 <code>isCopyNeeded()</code>	314
5.42.3.29 <code>makeCopy()</code>	314
5.43 <code>Tang::ComputedExpressionNativeBoundFunction</code> Class Reference	315
5.43.1 Detailed Description	317
5.43.2 Constructor & Destructor Documentation	317
5.43.2.1 <code>ComputedExpressionNativeBoundFunction()</code>	318
5.43.3 Member Function Documentation	318
5.43.3.1 <code>__add()</code>	318
5.43.3.2 <code>__asCode()</code>	319
5.43.3.3 <code>__assign_index()</code>	319
5.43.3.4 <code>__boolean()</code>	320
5.43.3.5 <code>__divide()</code>	320
5.43.3.6 <code>__equal()</code>	320
5.43.3.7 <code>__float()</code>	321
5.43.3.8 <code>__getIterator()</code>	321
5.43.3.9 <code>__index()</code>	321
5.43.3.10 <code>__integer()</code>	322
5.43.3.11 <code>__iteratorNext()</code>	322
5.43.3.12 <code>__lessThan()</code>	322
5.43.3.13 <code>__modulo()</code>	324
5.43.3.14 <code>__multiply()</code>	324
5.43.3.15 <code>__negative()</code>	325
5.43.3.16 <code>__not()</code>	325
5.43.3.17 <code>__period()</code>	325
5.43.3.18 <code>__slice()</code>	326
5.43.3.19 <code>__string()</code>	326
5.43.3.20 <code>__subtract()</code>	326
5.43.3.21 <code>dump()</code>	327
5.43.3.22 <code>getArgc()</code>	327
5.43.3.23 <code>getFunction()</code>	327

5.43.3.24 <code>getTargetTypeIndex()</code>	328
5.43.3.25 <code>is_equal()</code> [1/6]	328
5.43.3.26 <code>is_equal()</code> [2/6]	328
5.43.3.27 <code>is_equal()</code> [3/6]	329
5.43.3.28 <code>is_equal()</code> [4/6]	329
5.43.3.29 <code>is_equal()</code> [5/6]	329
5.43.3.30 <code>is_equal()</code> [6/6]	330
5.43.3.31 <code>isCopyNeeded()</code>	330
5.43.3.32 <code>makeCopy()</code>	330
5.44 <code>Tang::ComputedExpressionNativeFunction</code> Class Reference	331
5.44.1 Detailed Description	333
5.44.2 Constructor & Destructor Documentation	333
5.44.2.1 <code>ComputedExpressionNativeFunction()</code>	333
5.44.3 Member Function Documentation	333
5.44.3.1 <code>__add()</code>	334
5.44.3.2 <code>__asCode()</code>	334
5.44.3.3 <code>__assign_index()</code>	334
5.44.3.4 <code>__boolean()</code>	335
5.44.3.5 <code>__divide()</code>	335
5.44.3.6 <code>__equal()</code>	335
5.44.3.7 <code>__float()</code>	336
5.44.3.8 <code>__getIterator()</code>	336
5.44.3.9 <code>__index()</code>	337
5.44.3.10 <code>__integer()</code>	337
5.44.3.11 <code>__iteratorNext()</code>	337
5.44.3.12 <code>__lessThan()</code>	338
5.44.3.13 <code>__modulo()</code>	338
5.44.3.14 <code>__multiply()</code>	338
5.44.3.15 <code>__negative()</code>	339
5.44.3.16 <code>__not()</code>	339
5.44.3.17 <code>__period()</code>	339
5.44.3.18 <code>__slice()</code>	340
5.44.3.19 <code>__string()</code>	340
5.44.3.20 <code>__subtract()</code>	341
5.44.3.21 <code>dump()</code>	341
5.44.3.22 <code>getArgc()</code>	341
5.44.3.23 <code>getFunction()</code>	342
5.44.3.24 <code>is_equal()</code> [1/6]	342
5.44.3.25 <code>is_equal()</code> [2/6]	343
5.44.3.26 <code>is_equal()</code> [3/6]	343
5.44.3.27 <code>is_equal()</code> [4/6]	344
5.44.3.28 <code>is_equal()</code> [5/6]	344

5.44.3.29 <code>is_equal()</code> [6/6]	344
5.44.3.30 <code>isCopyNeeded()</code>	345
5.44.3.31 <code>makeCopy()</code>	345
5.45 <code>Tang::ComputedExpressionNativeLibraryFunction</code> Class Reference	346
5.45.1 Detailed Description	348
5.45.2 Constructor & Destructor Documentation	348
5.45.2.1 <code>ComputedExpressionNativeLibraryFunction()</code>	348
5.45.3 Member Function Documentation	348
5.45.3.1 <code>__add()</code>	348
5.45.3.2 <code>__asCode()</code>	349
5.45.3.3 <code>__assign_index()</code>	349
5.45.3.4 <code>__boolean()</code>	350
5.45.3.5 <code>__divide()</code>	350
5.45.3.6 <code>__equal()</code>	350
5.45.3.7 <code>__float()</code>	351
5.45.3.8 <code>__getIterator()</code>	351
5.45.3.9 <code>__index()</code>	351
5.45.3.10 <code>__integer()</code>	352
5.45.3.11 <code>__iteratorNext()</code>	352
5.45.3.12 <code>__lessThan()</code>	352
5.45.3.13 <code>__modulo()</code>	354
5.45.3.14 <code>__multiply()</code>	354
5.45.3.15 <code>__negative()</code>	355
5.45.3.16 <code>__not()</code>	355
5.45.3.17 <code>__period()</code>	355
5.45.3.18 <code>__slice()</code>	356
5.45.3.19 <code>__string()</code>	356
5.45.3.20 <code>__subtract()</code>	356
5.45.3.21 <code>dump()</code>	357
5.45.3.22 <code>getFunction()</code>	357
5.45.3.23 <code>is_equal()</code> [1/6]	357
5.45.3.24 <code>is_equal()</code> [2/6]	358
5.45.3.25 <code>is_equal()</code> [3/6]	358
5.45.3.26 <code>is_equal()</code> [4/6]	358
5.45.3.27 <code>is_equal()</code> [5/6]	359
5.45.3.28 <code>is_equal()</code> [6/6]	359
5.45.3.29 <code>isCopyNeeded()</code>	360
5.45.3.30 <code>makeCopy()</code>	360
5.46 <code>Tang::ComputedExpressionString</code> Class Reference	360
5.46.1 Detailed Description	363
5.46.2 Constructor & Destructor Documentation	363
5.46.2.1 <code>ComputedExpressionString()</code> [1/2]	363

5.46.2.2 <code>ComputedExpressionString()</code> [2/2]	363
5.46.3 Member Function Documentation	364
5.46.3.1 <code>__add()</code>	364
5.46.3.2 <code>__asCode()</code>	364
5.46.3.3 <code>__assign_index()</code>	365
5.46.3.4 <code>__boolean()</code>	365
5.46.3.5 <code>__divide()</code>	366
5.46.3.6 <code>__equal()</code>	366
5.46.3.7 <code>__float()</code>	367
5.46.3.8 <code>__getIterator()</code>	367
5.46.3.9 <code>__index()</code>	368
5.46.3.10 <code>__integer()</code>	368
5.46.3.11 <code>__iteratorNext()</code>	369
5.46.3.12 <code>__lessThan()</code>	369
5.46.3.13 <code>__modulo()</code>	370
5.46.3.14 <code>__multiply()</code>	370
5.46.3.15 <code>__negative()</code>	371
5.46.3.16 <code>__not()</code>	371
5.46.3.17 <code>__period()</code>	371
5.46.3.18 <code>__slice()</code>	372
5.46.3.19 <code>__string()</code>	373
5.46.3.20 <code>__subtract()</code>	373
5.46.3.21 <code>bytesLength()</code>	373
5.46.3.22 <code>dump()</code>	374
5.46.3.23 <code>getMethods()</code>	374
5.46.3.24 <code>getValue()</code>	374
5.46.3.25 <code>is_equal()</code> [1/6]	374
5.46.3.26 <code>is_equal()</code> [2/6]	375
5.46.3.27 <code>is_equal()</code> [3/6]	375
5.46.3.28 <code>is_equal()</code> [4/6]	376
5.46.3.29 <code>is_equal()</code> [5/6]	376
5.46.3.30 <code>is_equal()</code> [6/6]	377
5.46.3.31 <code>isCopyNeeded()</code>	377
5.46.3.32 <code>length()</code>	378
5.46.3.33 <code>makeCopy()</code>	378
5.46.3.34 <code>operator+=()</code>	378
5.47 <code>Tang::Context</code> Class Reference	379
5.47.1 Detailed Description	379
5.48 <code>Tang::Error</code> Class Reference	380
5.48.1 Detailed Description	381
5.48.2 Constructor & Destructor Documentation	381
5.48.2.1 <code>Error()</code> [1/2]	381

5.48.2.2 <code>Error()</code> [2/2]	381
5.48.3 Friends And Related Function Documentation	381
5.48.3.1 <code>operator<<</code>	382
5.49 <code>Tang::GarbageCollected</code> Class Reference	382
5.49.1 Detailed Description	384
5.49.2 Constructor & Destructor Documentation	384
5.49.2.1 <code>GarbageCollected()</code> [1/3]	384
5.49.2.2 <code>GarbageCollected()</code> [2/3]	385
5.49.2.3 <code>~GarbageCollected()</code>	385
5.49.2.4 <code>GarbageCollected()</code> [3/3]	385
5.49.3 Member Function Documentation	385
5.49.3.1 <code>isCopyNeeded()</code>	385
5.49.3.2 <code>make()</code>	386
5.49.3.3 <code>makeCopy()</code>	386
5.49.3.4 <code>operator"!"()</code>	387
5.49.3.5 <code>operator"!=()</code>	387
5.49.3.6 <code>operator%()</code>	388
5.49.3.7 <code>operator*()</code> [1/2]	389
5.49.3.8 <code>operator*()</code> [2/2]	389
5.49.3.9 <code>operator+()</code>	389
5.49.3.10 <code>operator-()</code> [1/2]	390
5.49.3.11 <code>operator-()</code> [2/2]	390
5.49.3.12 <code>operator->()</code>	391
5.49.3.13 <code>operator/()</code>	391
5.49.3.14 <code>operator<()</code>	392
5.49.3.15 <code>operator<=()</code>	392
5.49.3.16 <code>operator=()</code> [1/2]	393
5.49.3.17 <code>operator=()</code> [2/2]	393
5.49.3.18 <code>operator==()</code> [1/8]	393
5.49.3.19 <code>operator==()</code> [2/8]	394
5.49.3.20 <code>operator==()</code> [3/8]	394
5.49.3.21 <code>operator==()</code> [4/8]	394
5.49.3.22 <code>operator==()</code> [5/8]	395
5.49.3.23 <code>operator==()</code> [6/8]	395
5.49.3.24 <code>operator==()</code> [7/8]	396
5.49.3.25 <code>operator==()</code> [8/8]	396
5.49.3.26 <code>operator>()</code>	397
5.49.3.27 <code>operator>=()</code>	397
5.49.4 Friends And Related Function Documentation	397
5.49.4.1 <code>operator<<</code>	398
5.50 <code>Tang::HtmlEscape</code> Class Reference	398
5.50.1 Detailed Description	399

5.50.2 Constructor & Destructor Documentation	399
5.50.2.1 HtmlEscape()	399
5.50.3 Member Function Documentation	400
5.50.3.1 get_next_token()	400
5.51 Tang::HtmlEscapeAscii Class Reference	400
5.51.1 Detailed Description	401
5.51.2 Constructor & Destructor Documentation	401
5.51.2.1 HtmlEscapeAscii()	401
5.51.3 Member Function Documentation	402
5.51.3.1 get_next_token()	402
5.52 Tang::location Class Reference	402
5.52.1 Detailed Description	403
5.53 Tang::position Class Reference	404
5.53.1 Detailed Description	405
5.54 Tang::Program Class Reference	405
5.54.1 Detailed Description	408
5.54.2 Member Enumeration Documentation	408
5.54.2.1 CodeType	408
5.54.3 Constructor & Destructor Documentation	408
5.54.3.1 Program()	408
5.54.4 Member Function Documentation	408
5.54.4.1 addBreak()	409
5.54.4.2 addBytecode()	409
5.54.4.3 addContinue()	409
5.54.4.4 addIdentifier()	409
5.54.4.5 addIdentifierAssigned()	410
5.54.4.6 addLibraryAlias()	410
5.54.4.7 addString()	410
5.54.4.8 dumpBytecode()	411
5.54.4.9 execute() [1/2]	411
5.54.4.10 execute() [2/2]	411
5.54.4.11 getAst()	412
5.54.4.12 getBytecode()	412
5.54.4.13 getCode()	413
5.54.4.14 getIdentifiers()	413
5.54.4.15 getIdentifiersAssigned()	413
5.54.4.16 getLibraryAliases()	413
5.54.4.17 getResult()	414
5.54.4.18 getStrings()	414
5.54.4.19 popBreakStack()	414
5.54.4.20 popContinueStack()	415
5.54.4.21 pushEnvironment()	415

5.54.4.22 setFunctionStackDeclaration()	416
5.54.4.23 setJumpTarget()	416
5.54.5 Member Data Documentation	416
5.54.5.1 functionsDeclared	416
5.55 Tang::SingletonObjectPool< T > Class Template Reference	417
5.55.1 Detailed Description	418
5.55.2 Member Function Documentation	418
5.55.2.1 get()	418
5.55.2.2 getInstance()	419
5.55.2.3 recycle()	419
5.55.3 Member Data Documentation	419
5.55.3.1 currentIndex	419
5.55.3.2 currentRecycledIndex	419
5.56 Tang::TangBase Class Reference	420
5.56.1 Detailed Description	421
5.56.2 Constructor & Destructor Documentation	421
5.56.2.1 TangBase()	421
5.56.3 Member Function Documentation	422
5.56.3.1 compileScript()	422
5.56.3.2 compileTemplate()	422
5.56.3.3 make_shared()	422
5.57 Tang::TangScanner Class Reference	423
5.57.1 Detailed Description	424
5.57.2 Constructor & Destructor Documentation	424
5.57.2.1 TangScanner()	424
5.57.3 Member Function Documentation	425
5.57.3.1 get_next_token()	425
5.58 Tang::Unescape Class Reference	425
5.58.1 Detailed Description	426
5.58.2 Constructor & Destructor Documentation	426
5.58.2.1 Unescape()	426
5.58.3 Member Function Documentation	426
5.58.3.1 get_next_token()	426
5.59 Tang::UnicodeString Class Reference	427
5.59.1 Detailed Description	428
5.59.2 Member Enumeration Documentation	428
5.59.2.1 Type	428
5.59.3 Member Function Documentation	429
5.59.3.1 bytesLength()	429
5.59.3.2 length()	429
5.59.3.3 operator std::string()	430
5.59.3.4 operator+()	430

5.59.3.5 operator+=()	430
5.59.3.6 operator<()	431
5.59.3.7 operator==()	431
5.59.3.8 render()	431
5.59.3.9 renderAscii()	432
5.59.3.10 substr()	432
6 File Documentation	435
6.1 build/generated/location.hh File Reference	435
6.1.1 Detailed Description	436
6.1.2 Function Documentation	436
6.1.2.1 operator<<() [1/2]	436
6.1.2.2 operator<<() [2/2]	437
6.2 include/astNode.hpp File Reference	437
6.2.1 Detailed Description	438
6.3 include/astNodeArray.hpp File Reference	438
6.3.1 Detailed Description	439
6.4 include/astNodeAssign.hpp File Reference	439
6.4.1 Detailed Description	440
6.5 include/astNodeBinary.hpp File Reference	440
6.5.1 Detailed Description	440
6.6 include/astNodeBlock.hpp File Reference	441
6.6.1 Detailed Description	441
6.7 include/astNodeBoolean.hpp File Reference	442
6.7.1 Detailed Description	442
6.8 include/astNodeBreak.hpp File Reference	443
6.8.1 Detailed Description	443
6.9 include/astNodeCast.hpp File Reference	444
6.9.1 Detailed Description	444
6.10 include/astNodeContinue.hpp File Reference	445
6.10.1 Detailed Description	445
6.11 include/astNodeDoWhile.hpp File Reference	446
6.11.1 Detailed Description	446
6.12 include/astNodeFloat.hpp File Reference	447
6.12.1 Detailed Description	447
6.13 include/astNodeFor.hpp File Reference	448
6.13.1 Detailed Description	448
6.14 include/astNodeFunctionCall.hpp File Reference	449
6.14.1 Detailed Description	449
6.15 include/astNodeFunctionDeclaration.hpp File Reference	450
6.15.1 Detailed Description	450
6.16 include/astNodeIdentifier.hpp File Reference	451

6.16.1 Detailed Description	451
6.17 include/astNodeIfElse.hpp File Reference	452
6.17.1 Detailed Description	452
6.18 include/astNodeIndex.hpp File Reference	453
6.18.1 Detailed Description	453
6.19 include/astNodeInteger.hpp File Reference	454
6.19.1 Detailed Description	454
6.20 include/astNodeLibrary.hpp File Reference	455
6.20.1 Detailed Description	455
6.21 include/astNodeMap.hpp File Reference	456
6.21.1 Detailed Description	456
6.22 include/astNodePeriod.hpp File Reference	457
6.22.1 Detailed Description	457
6.23 include/astNodePrint.hpp File Reference	458
6.23.1 Detailed Description	458
6.24 include/astNodeRangedFor.hpp File Reference	459
6.24.1 Detailed Description	459
6.25 include/astNodeReturn.hpp File Reference	460
6.25.1 Detailed Description	460
6.26 include/astNodeSlice.hpp File Reference	461
6.26.1 Detailed Description	461
6.27 include/astNodeString.hpp File Reference	462
6.27.1 Detailed Description	462
6.28 include/astNodeTernary.hpp File Reference	462
6.28.1 Detailed Description	463
6.29 include/astNodeUnary.hpp File Reference	463
6.29.1 Detailed Description	464
6.30 include/astNodeUse.hpp File Reference	464
6.30.1 Detailed Description	465
6.31 include/astNodeWhile.hpp File Reference	465
6.31.1 Detailed Description	466
6.32 include/computedExpression.hpp File Reference	466
6.32.1 Detailed Description	467
6.33 include/computedExpressionArray.hpp File Reference	467
6.33.1 Detailed Description	468
6.34 include/computedExpressionBoolean.hpp File Reference	468
6.34.1 Detailed Description	469
6.35 include/computedExpressionCompiledFunction.hpp File Reference	469
6.35.1 Detailed Description	470
6.36 include/computedExpressionError.hpp File Reference	470
6.36.1 Detailed Description	471
6.37 include/computedExpressionFloat.hpp File Reference	471

6.37.1 Detailed Description	471
6.38 include/computedExpressionInteger.hpp File Reference	472
6.38.1 Detailed Description	472
6.39 include/computedExpressionIterator.hpp File Reference	472
6.39.1 Detailed Description	473
6.40 include/computedExpressionIteratorEnd.hpp File Reference	473
6.40.1 Detailed Description	474
6.41 include/computedExpressionLibrary.hpp File Reference	474
6.41.1 Detailed Description	475
6.42 include/computedExpressionLibraryMath.hpp File Reference	475
6.42.1 Detailed Description	476
6.43 include/computedExpressionMap.hpp File Reference	477
6.43.1 Detailed Description	477
6.44 include/computedExpressionNativeBoundFunction.hpp File Reference	478
6.44.1 Detailed Description	478
6.45 include/computedExpressionNativeFunction.hpp File Reference	479
6.45.1 Detailed Description	479
6.46 include/computedExpressionNativeLibraryFunction.hpp File Reference	480
6.46.1 Detailed Description	480
6.47 include/computedExpressionString.hpp File Reference	481
6.47.1 Detailed Description	481
6.48 include/context.hpp File Reference	481
6.48.1 Detailed Description	482
6.49 include/error.hpp File Reference	482
6.49.1 Detailed Description	483
6.50 include/garbageCollected.hpp File Reference	483
6.50.1 Detailed Description	484
6.51 include/htmlEscape.hpp File Reference	484
6.51.1 Detailed Description	485
6.52 include/htmlEscapeAscii.hpp File Reference	485
6.52.1 Detailed Description	486
6.53 include/macros.hpp File Reference	487
6.53.1 Detailed Description	488
6.54 include/opcode.hpp File Reference	488
6.54.1 Detailed Description	488
6.54.2 Enumeration Type Documentation	488
6.54.2.1 Opcode	488
6.55 include/program.hpp File Reference	490
6.55.1 Detailed Description	490
6.56 include/singleObjectPool.hpp File Reference	491
6.56.1 Detailed Description	491
6.57 include/tang.hpp File Reference	492

6.57.1 Detailed Description	492
6.58 include/tangBase.hpp File Reference	493
6.58.1 Detailed Description	493
6.59 include/tangScanner.hpp File Reference	494
6.59.1 Detailed Description	495
6.60 include/unescape.hpp File Reference	495
6.60.1 Detailed Description	496
6.61 include/unicodeString.hpp File Reference	496
6.61.1 Detailed Description	497
6.61.2 Function Documentation	497
6.61.2.1 htmlEscape()	497
6.61.2.2 htmlEscapeAscii()	497
6.61.2.3 unescape()	498
6.62 src/astNode.cpp File Reference	499
6.62.1 Detailed Description	499
6.63 src/astNodeArray.cpp File Reference	499
6.63.1 Detailed Description	500
6.64 src/astNodeAssign.cpp File Reference	500
6.64.1 Detailed Description	500
6.65 src/astNodeBinary.cpp File Reference	500
6.65.1 Detailed Description	501
6.66 src/astNodeBlock.cpp File Reference	501
6.66.1 Detailed Description	501
6.67 src/astNodeBoolean.cpp File Reference	502
6.67.1 Detailed Description	502
6.68 src/astNodeBreak.cpp File Reference	502
6.68.1 Detailed Description	503
6.69 src/astNodeCast.cpp File Reference	503
6.69.1 Detailed Description	503
6.70 src/astNodeContinue.cpp File Reference	503
6.70.1 Detailed Description	504
6.71 src/astNodeDoWhile.cpp File Reference	504
6.71.1 Detailed Description	504
6.72 src/astNodeFloat.cpp File Reference	504
6.72.1 Detailed Description	505
6.73 src/astNodeFor.cpp File Reference	505
6.73.1 Detailed Description	505
6.74 src/astNodeFunctionCall.cpp File Reference	505
6.74.1 Detailed Description	506
6.75 src/astNodeFunctionDeclaration.cpp File Reference	506
6.75.1 Detailed Description	506
6.76 src/astNodeIdentifier.cpp File Reference	506

6.76.1 Detailed Description	507
6.77 src/astNodeIfElse.cpp File Reference	507
6.77.1 Detailed Description	507
6.78 src/astNodeIndex.cpp File Reference	508
6.78.1 Detailed Description	508
6.79 src/astNodeInteger.cpp File Reference	508
6.79.1 Detailed Description	509
6.80 src/astNodeLibrary.cpp File Reference	509
6.80.1 Detailed Description	509
6.81 src/astNodeMap.cpp File Reference	509
6.81.1 Detailed Description	510
6.82 src/astNodePeriod.cpp File Reference	510
6.82.1 Detailed Description	510
6.83 src/astNodePrint.cpp File Reference	511
6.83.1 Detailed Description	511
6.84 src/astNodeRangedFor.cpp File Reference	511
6.84.1 Detailed Description	512
6.85 src/astNodeReturn.cpp File Reference	512
6.85.1 Detailed Description	512
6.86 src/astNodeSlice.cpp File Reference	512
6.86.1 Detailed Description	513
6.87 src/astNodeString.cpp File Reference	513
6.87.1 Detailed Description	513
6.88 src/astNodeTernary.cpp File Reference	513
6.88.1 Detailed Description	514
6.89 src/astNodeUnary.cpp File Reference	514
6.89.1 Detailed Description	514
6.90 src/astNodeUse.cpp File Reference	514
6.90.1 Detailed Description	515
6.91 src/astNodeWhile.cpp File Reference	515
6.91.1 Detailed Description	515
6.92 src/computedExpression.cpp File Reference	515
6.92.1 Detailed Description	516
6.93 src/computedExpressionArray.cpp File Reference	516
6.93.1 Detailed Description	516
6.94 src/computedExpressionBoolean.cpp File Reference	517
6.94.1 Detailed Description	517
6.95 src/computedExpressionCompiledFunction.cpp File Reference	517
6.95.1 Detailed Description	518
6.96 src/computedExpressionError.cpp File Reference	518
6.96.1 Detailed Description	518
6.97 src/computedExpressionFloat.cpp File Reference	518

6.97.1 Detailed Description	519
6.98 src/computedExpressionInteger.cpp File Reference	519
6.98.1 Detailed Description	519
6.99 src/computedExpressionIterator.cpp File Reference	520
6.99.1 Detailed Description	520
6.100 src/computedExpressionIteratorEnd.cpp File Reference	520
6.100.1 Detailed Description	521
6.101 src/computedExpressionLibrary.cpp File Reference	521
6.101.1 Detailed Description	521
6.102 src/computedExpressionLibraryMath.cpp File Reference	521
6.102.1 Detailed Description	522
6.103 src/computedExpressionMap.cpp File Reference	522
6.103.1 Detailed Description	522
6.104 src/computedExpressionNativeBoundFunction.cpp File Reference	523
6.104.1 Detailed Description	523
6.105 src/computedExpressionNativeFunction.cpp File Reference	523
6.105.1 Detailed Description	524
6.106 src/computedExpressionNativeLibraryFunction.cpp File Reference	524
6.106.1 Detailed Description	524
6.107 src/computedExpressionString.cpp File Reference	524
6.107.1 Detailed Description	525
6.108 src/context.cpp File Reference	525
6.108.1 Detailed Description	525
6.109 src/error.cpp File Reference	526
6.109.1 Detailed Description	526
6.109.2 Function Documentation	526
6.109.2.1 operator<<()	526
6.110 src/garbageCollected.cpp File Reference	527
6.110.1 Function Documentation	527
6.110.1.1 operator<<()	527
6.111 src/program-dumpBytecode.cpp File Reference	528
6.111.1 Detailed Description	528
6.111.2 Macro Definition Documentation	528
6.111.2.1 DUMPPROGRAMCHECK	529
6.112 src/program-execute.cpp File Reference	529
6.112.1 Detailed Description	530
6.112.2 Macro Definition Documentation	530
6.112.2.1 EXECUTEPROGRAMCHECK	530
6.112.2.2 STACKCHECK	530
6.113 src/program.cpp File Reference	531
6.113.1 Detailed Description	531
6.114 src/tangBase.cpp File Reference	531

6.114.1 Detailed Description	532
6.115 src/unicodeString.cpp File Reference	532
6.115.1 Detailed Description	532
6.116 test/test.cpp File Reference	533
6.116.1 Detailed Description	534
6.117 test/testGarbageCollected.cpp File Reference	534
6.117.1 Detailed Description	535
6.118 test/testSingletonObjectPool.cpp File Reference	535
6.118.1 Detailed Description	536
6.119 test/testUnicodeString.cpp File Reference	536
6.119.1 Detailed Description	536
Index	537

Chapter 1

Tang: A Template Language

1.1 Quick Description

Tang is a C++ Template Language. It takes the form of a library which may be included in other projects. It is under active development, and you can follow its progress here:

- [YouTube playlist](#)
- [GitHub repository](#)

1.2 Features

The following features are planned:

- Native support for Unicode/Utf-8 strings.
- Change from template to script mode using escape tags like PHP.
- Loosely typed, with Python-like indexing and slicing of containers.
- Syntax similar to C/C++/PHP.
- Code compiles to a custom Bytecode and is executed by the Tang VM.
- Fast and thread-safe.

1.3 License

MIT License

Copyright (c) 2022 Corey Pennycuff

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Tang::AstNode	15
Tang::AstNodeArray	21
Tang::AstNodeAssign	25
Tang::AstNodeBinary	29
Tang::AstNodeBlock	35
Tang::AstNodeBoolean	39
Tang::AstNodeBreak	43
Tang::AstNodeCast	47
Tang::AstNodeContinue	52
Tang::AstNodeDoWhile	56
Tang::AstNodeFloat	60
Tang::AstNodeFor	64
Tang::AstNodeFunctionCall	69
Tang::AstNodeFunctionDeclaration	72
Tang::AstNodeIdentifier	76
Tang::AstNodeIfElse	81
Tang::AstNodeIndex	86
Tang::AstNodeInteger	91
Tang::AstNodeLibrary	95
Tang::AstNodeMap	99
Tang::AstNodePeriod	103
Tang::AstNodePrint	107
Tang::AstNodeRangedFor	112
Tang::AstNodeReturn	116
Tang::AstNodeSlice	120
Tang::AstNodeString	125
Tang::AstNodeTernary	130
Tang::AstNodeUnary	135
Tang::AstNodeUse	141
Tang::AstNodeWhile	145
Tang::ComputedExpression	150
Tang::ComputedExpressionArray	163
Tang::ComputedExpressionBoolean	179
Tang::ComputedExpressionCompiledFunction	191
Tang::ComputedExpressionError	204

Tang::ComputedExpressionFloat	217
Tang::ComputedExpressionInteger	231
Tang::ComputedExpressionIterator	246
Tang::ComputedExpressionIteratorEnd	260
Tang::ComputedExpressionLibrary	274
Tang::ComputedExpressionLibraryMath	288
Tang::ComputedExpressionMap	301
Tang::ComputedExpressionNativeBoundFunction	315
Tang::ComputedExpressionNativeFunction	331
Tang::ComputedExpressionNativeLibraryFunction	346
Tang::ComputedExpressionString	360
Tang::Context	379
std::enable_shared_from_this	
Tang::TangBase	420
Tang::Error	380
Tang::GarbageCollected	382
Tang::location	402
Tang::position	404
Tang::Program	405
Tang::SingletonObjectPool< T >	417
TangHtmlEscapeAsciiFlexLexer	
Tang::HtmlEscapeAscii	400
TangHtmlEscapeFlexLexer	
Tang::HtmlEscape	398
TangTangFlexLexer	
Tang::TangScanner	423
TangUnescapeFlexLexer	
Tang::Unescape	425
Tang::UnicodeString	427

Chapter 3

Class Index

3.1 Class List

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

Tang::AstNode	Base class for representing nodes of an Abstract Syntax Tree (AST)	15
Tang::AstNodeArray	An AstNode that represents an array literal	21
Tang::AstNodeAssign	An AstNode that represents a binary expression	25
Tang::AstNodeBinary	An AstNode that represents a binary expression	29
Tang::AstNodeBlock	An AstNode that represents a code block	35
Tang::AstNodeBoolean	An AstNode that represents a boolean literal	39
Tang::AstNodeBreak	An AstNode that represents a <code>break</code> statement	43
Tang::AstNodeCast	An AstNode that represents a typecast of an expression	47
Tang::AstNodeContinue	An AstNode that represents a <code>continue</code> statement	52
Tang::AstNodeDoWhile	An AstNode that represents a <code>do..while</code> statement	56
Tang::AstNodeFloat	An AstNode that represents an float literal	60
Tang::AstNodeFor	An AstNode that represents an <code>if()</code> statement	64
Tang::AstNodeFunctionCall	An AstNode that represents a function call	69
Tang::AstNodeFunctionDeclaration	An AstNode that represents a function declaration	72
Tang::AstNodeIdentifier	An AstNode that represents an identifier	76
Tang::AstNodeIfElse	An AstNode that represents an <code>if..else</code> statement	81
Tang::AstNodeIndex	An AstNode that represents an index into a collection	86
Tang::AstNodeInteger	An AstNode that represents an integer literal	91

Tang::AstNodeLibrary	An <code>AstNode</code> that represents an identifier	95
Tang::AstNodeMap	An <code>AstNode</code> that represents a map literal	99
Tang::AstNodePeriod	An <code>AstNode</code> that represents a member access (period) into an object	103
Tang::AstNodePrint	An <code>AstNode</code> that represents a print typeeration	107
Tang::AstNodeRangedFor	An <code>AstNode</code> that represents a ranged for() statement	112
Tang::AstNodeReturn	An <code>AstNode</code> that represents a <code>return</code> statement	116
Tang::AstNodeSlice	An <code>AstNode</code> that represents a ternary expression	120
Tang::AstNodeString	An <code>AstNode</code> that represents a string literal	125
Tang::AstNodeTernary	An <code>AstNode</code> that represents a ternary expression	130
Tang::AstNodeUnary	An <code>AstNode</code> that represents a unary negation	135
Tang::AstNodeUse	An <code>AstNode</code> that represents the inclusion of a library into the script	141
Tang::AstNodeWhile	An <code>AstNode</code> that represents a while statement	145
Tang::ComputedExpression	Represents the result of a computation that has been executed	150
Tang::ComputedExpressionArray	Represents an Array that is the result of a computation	163
Tang::ComputedExpressionBoolean	Represents an Boolean that is the result of a computation	179
Tang::ComputedExpressionCompiledFunction	Represents a Compiled Function declared in the script	191
Tang::ComputedExpressionError	Represents a Runtime <code>Error</code>	204
Tang::ComputedExpressionFloat	Represents a Float that is the result of a computation	217
Tang::ComputedExpressionInteger	Represents an Integer that is the result of a computation	231
Tang::ComputedExpressionIterator	Represents an Iterator that is the result of a computation	246
Tang::ComputedExpressionIteratorEnd	Represents that a collection has no more values through which to iterate	260
Tang::ComputedExpressionLibrary	Represents a Runtime Library	274
Tang::ComputedExpressionLibraryMath	Represents a Runtime LibraryMath	288
Tang::ComputedExpressionMap	Represents an Map that is the result of a computation	301
Tang::ComputedExpressionNativeBoundFunction	Represents a NativeBound Function declared in the script	315
Tang::ComputedExpressionNativeFunction	Represents a Native Function provided by compiled C++ code	331
Tang::ComputedExpressionNativeLibraryFunction	Represents a Native Function provided by compiled C++ code that is executed to create a library or one of its attributes	346
Tang::ComputedExpressionString	Represents a String that is the result of a computation	360

Tang::Context	Holds all environment variables specific to the execution of a program	379
Tang::Error	Used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error	380
Tang::GarbageCollected	A container that acts as a resource-counting garbage collector for the specified type	382
Tang::HtmlEscape	The Flex lexer class for the main Tang language	398
Tang::HtmlEscapeAscii	The Flex lexer class for the main Tang language	400
Tang::location	Two points in a source file	402
Tang::position	A point in a source file	404
Tang::Program	Represents a compiled script or template that may be executed	405
Tang::SingletonObjectPool< T >	A thread-safe, singleton object pool of the designated type	417
Tang::TangBase	The base class for the Tang programming language	420
Tang::TangScanner	The Flex lexer class for the main Tang language	423
Tang::Unescape	The Flex lexer class for the main Tang language	425
Tang::UnicodeString	Represents a UTF-8 encoded string that is Unicode-aware	427

Chapter 4

File Index

4.1 File List

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

build/generated/ location.hh	Define the Tang ::location class	435
include/ astNode.hpp	Declare the Tang::AstNode base class	437
include/ astNodeArray.hpp	Declare the Tang::AstNodeArray class	438
include/ astNodeAssign.hpp	Declare the Tang::AstNodeAssign class	439
include/ astNodeBinary.hpp	Declare the Tang::AstNodeBinary class	440
include/ astNodeBlock.hpp	Declare the Tang::AstNodeBlock class	441
include/ astNodeBoolean.hpp	Declare the Tang::AstNodeBoolean class	442
include/ astNodeBreak.hpp	Declare the Tang::AstNodeBreak class	443
include/ astNodeCast.hpp	Declare the Tang::AstNodeCast class	444
include/ astNodeContinue.hpp	Declare the Tang::AstNodeContinue class	445
include/ astNodeDoWhile.hpp	Declare the Tang::AstNodeDoWhile class	446
include/ astNodeFloat.hpp	Declare the Tang::AstNodeFloat class	447
include/ astNodeFor.hpp	Declare the Tang::AstNodeFor class	448
include/ astNodeFunctionCall.hpp	Declare the Tang::AstNodeFunctionCall class	449
include/ astNodeFunctionDeclaration.hpp	Declare the Tang::AstNodeFunctionDeclaration class	450
include/ astNodeIdentifier.hpp	Declare the Tang::AstNodeIdentifier class	451
include/ astNodeIfElse.hpp	Declare the Tang::AstNodeIfElse class	452
include/ astNodeIndex.hpp	Declare the Tang::AstNodeIndex class	453

include/astNodeInteger.hpp	Declare the <code>Tang::AstNodeInteger</code> class	454
include/astNodeLibrary.hpp	Declare the <code>Tang::AstNodeLibrary</code> class	455
include/astNodeMap.hpp	Declare the <code>Tang::AstNodeMap</code> class	456
include/astNodePeriod.hpp	Declare the <code>Tang::AstNodePeriod</code> class	457
include/astNodePrint.hpp	Declare the <code>Tang::AstNodePrint</code> class	458
include/astNodeRangedFor.hpp	Declare the <code>Tang::AstNodeRangedFor</code> class	459
include/astNodeReturn.hpp	Declare the <code>Tang::AstNodeReturn</code> class	460
include/astNodeSlice.hpp	Declare the <code>Tang::AstNodeSlice</code> class	461
include/astNodeString.hpp	Declare the <code>Tang::AstNodeString</code> class	462
include/astNodeTernary.hpp	Declare the <code>Tang::AstNodeTernary</code> class	462
include/astNodeUnary.hpp	Declare the <code>Tang::AstNodeUnary</code> class	463
include/astNodeUse.hpp	Declare the <code>Tang::AstNodeUse</code> class	464
include/astNodeWhile.hpp	Declare the <code>Tang::AstNodeWhile</code> class	465
include/computedExpression.hpp	Declare the <code>Tang::ComputedExpression</code> base class	466
include/computedExpressionArray.hpp	Declare the <code>Tang::ComputedExpressionArray</code> class	467
include/computedExpressionBoolean.hpp	Declare the <code>Tang::ComputedExpressionBoolean</code> class	468
include/computedExpressionCompiledFunction.hpp	Declare the <code>Tang::ComputedExpressionCompiledFunction</code> class	469
include/computedExpressionError.hpp	Declare the <code>Tang::ComputedExpressionError</code> class	470
include/computedExpressionFloat.hpp	Declare the <code>Tang::ComputedExpressionFloat</code> class	471
include/computedExpressionInteger.hpp	Declare the <code>Tang::ComputedExpressionInteger</code> class	472
include/computedExpressionIterator.hpp	Declare the <code>Tang::ComputedExpressionIterator</code> class	472
include/computedExpressionIteratorEnd.hpp	Declare the <code>Tang::ComputedExpressionIteratorEnd</code> class	473
include/computedExpressionLibrary.hpp	Declare the <code>Tang::ComputedExpressionLibrary</code> class	474
include/computedExpressionLibraryMath.hpp	Declare the <code>Tang::ComputedExpressionLibraryMath</code> class	475
include/computedExpressionMap.hpp	Declare the <code>Tang::ComputedExpressionMap</code> class	477
include/computedExpressionNativeBoundFunction.hpp	Declare the <code>Tang::ComputedExpressionNativeBoundFunction</code> class	478
include/computedExpressionNativeFunction.hpp	Declare the <code>Tang::ComputedExpressionNativeFunction</code> class	479
include/computedExpressionNativeLibraryFunction.hpp	Declare the <code>Tang::ComputedExpressionNativeLibraryFunction</code> class	480
include/computedExpressionString.hpp	Declare the <code>Tang::ComputedExpressionString</code> class	481

include/context.hpp	Declare the <code>Tang::Context</code> class	481
include/error.hpp	Declare the <code>Tang::Error</code> class used to describe syntax and runtime errors	482
include/garbageCollected.hpp	Declare the <code>Tang::GarbageCollected</code> class	483
include/htmlEscape.hpp	Declare the <code>Tang::HtmlEscape</code> used to tokenize a Tang script	484
include/htmlEscapeAscii.hpp	Declare the <code>Tang::HtmlEscapeAscii</code> used to tokenize a Tang script	485
include/macros.hpp	Contains generic macros	487
include/opcode.hpp	Declare the Opcodes used in the Bytecode representation of a program	488
include/program.hpp	Declare the <code>Tang::Program</code> class used to compile and execute source code	490
include/singletonObjectPool.hpp	Declare the <code>Tang::SingletonObjectPool</code> class	491
include/tang.hpp	Header file supplied for use by 3rd party code so that they can easily include all necessary headers	492
include/tangBase.hpp	Declare the <code>Tang::TangBase</code> class used to interact with Tang	493
include/tangScanner.hpp	Declare the <code>Tang::TangScanner</code> used to tokenize a Tang script	494
include/unescape.hpp	Declare the <code>Tang::Unescape</code> used to tokenize a Tang script	495
include/unicodeString.hpp	Contains the code to interface with the ICU library	496
src/astNode.cpp	Define the <code>Tang::AstNode</code> class	499
src/astNodeArray.cpp	Define the <code>Tang::AstNodeArray</code> class	499
src/astNodeAssign.cpp	Define the <code>Tang::AstNodeAssign</code> class	500
src/astNodeBinary.cpp	Define the <code>Tang::AstNodeBinary</code> class	500
src/astNodeBlock.cpp	Define the <code>Tang::AstNodeBlock</code> class	501
src/astNodeBoolean.cpp	Define the <code>Tang::AstNodeBoolean</code> class	502
src/astNodeBreak.cpp	Define the <code>Tang::AstNodeBreak</code> class	502
src/astNodeCast.cpp	Define the <code>Tang::AstNodeCast</code> class	503
src/astNodeContinue.cpp	Define the <code>Tang::AstNodeContinue</code> class	503
src/astNodeDoWhile.cpp	Define the <code>Tang::AstNodeDoWhile</code> class	504
src/astNodeFloat.cpp	Define the <code>Tang::AstNodeFloat</code> class	504
src/astNodeFor.cpp	Define the <code>Tang::AstNodeFor</code> class	505
src/astNodeFunctionCall.cpp	Define the <code>Tang::AstNodeFunctionCall</code> class	505
src/astNodeFunctionDeclaration.cpp	Define the <code>Tang::AstNodeFunctionDeclaration</code> class	506

src/astNodeIdentifier.cpp	Define the <code>Tang::AstNodeIdentifier</code> class	506
src/astNodeIfElse.cpp	Define the <code>Tang::AstNodeIfElse</code> class	507
src/astNodeIndex.cpp	Define the <code>Tang::AstNodeIndex</code> class	508
src/astNodeInteger.cpp	Define the <code>Tang::AstNodeInteger</code> class	508
src/astNodeLibrary.cpp	Define the <code>Tang::AstNodeLibrary</code> class	509
src/astNodeMap.cpp	Define the <code>Tang::AstNodeMap</code> class	509
src/astNodePeriod.cpp	Define the <code>Tang::AstNodePeriod</code> class	510
src/astNodePrint.cpp	Define the <code>Tang::AstNodePrint</code> class	511
src/astNodeRangedFor.cpp	Define the <code>Tang::AstNodeRangedFor</code> class	511
src/astNodeReturn.cpp	Define the <code>Tang::AstNodeReturn</code> class	512
src/astNodeSlice.cpp	Define the <code>Tang::AstNodeSlice</code> class	512
src/astNodeString.cpp	Define the <code>Tang::AstNodeString</code> class	513
src/astNodeTernary.cpp	Define the <code>Tang::AstNodeTernary</code> class	513
src/astNodeUnary.cpp	Define the <code>Tang::AstNodeUnary</code> class	514
src/astNodeUse.cpp	Define the <code>Tang::AstNodeUse</code> class	514
src/astNodeWhile.cpp	Define the <code>Tang::AstNodeWhile</code> class	515
src/computedExpression.cpp	Define the <code>Tang::ComputedExpression</code> class	515
src/computedExpressionArray.cpp	Define the <code>Tang::ComputedExpressionArray</code> class	516
src/computedExpressionBoolean.cpp	Define the <code>Tang::ComputedExpressionBoolean</code> class	517
src/computedExpressionCompiledFunction.cpp	Define the <code>Tang::ComputedExpressionCompiledFunction</code> class	517
src/computedExpressionError.cpp	Define the <code>Tang::ComputedExpressionError</code> class	518
src/computedExpressionFloat.cpp	Define the <code>Tang::ComputedExpressionFloat</code> class	518
src/computedExpressionInteger.cpp	Define the <code>Tang::ComputedExpressionInteger</code> class	519
src/computedExpressionIterator.cpp	Define the <code>Tang::ComputedExpressionIterator</code> class	520
src/computedExpressionIteratorEnd.cpp	Define the <code>Tang::ComputedExpressionIteratorEnd</code> class	520
src/computedExpressionLibrary.cpp	Define the <code>Tang::ComputedExpressionLibrary</code> class	521
src/computedExpressionLibraryMath.cpp	Define the <code>Tang::ComputedExpressionLibraryMath</code> class	521
src/computedExpressionMap.cpp	Define the <code>Tang::ComputedExpressionMap</code> class	522
src/computedExpressionNativeBoundFunction.cpp	Define the <code>Tang::ComputedExpressionNativeBoundFunction</code> class	523

src/computedExpressionNativeFunction.cpp	Define the <code>Tang::ComputedExpressionNativeFunction</code> class	523
src/computedExpressionNativeLibraryFunction.cpp	Define the <code>Tang::ComputedExpressionNativeLibraryFunction</code> class	524
src/computedExpressionString.cpp	Define the <code>Tang::ComputedExpressionString</code> class	524
src/context.cpp	Define the <code>Tang::Context</code> class	525
src/error.cpp	Define the <code>Tang::Error</code> class	526
src/garbageCollected.cpp	527
src/program-dumpBytecode.cpp	Define the <code>Tang::Program::dumpBytecode</code> method	528
src/program-execute.cpp	Define the <code>Tang::Program::execute</code> method	529
src/program.cpp	Define the <code>Tang::Program</code> class	531
src/tangBase.cpp	Define the <code>Tang::TangBase</code> class	531
src/unicodeString.cpp	Contains the function declarations for the <code>Tang::UnicodeString</code> class and the interface to ICU	532
test/test.cpp	Test the general language behaviors	533
test/testGarbageCollected.cpp	Test the generic behavior of the <code>Tang::GarbageCollected</code> class	534
test/testSingletonObjectPool.cpp	Test the generic behavior of the <code>Tang::SingletonObjectPool</code> class	535
test/testUnicodeString.cpp	Contains tests for the <code>Tang::UnicodeString</code> class	536

Chapter 5

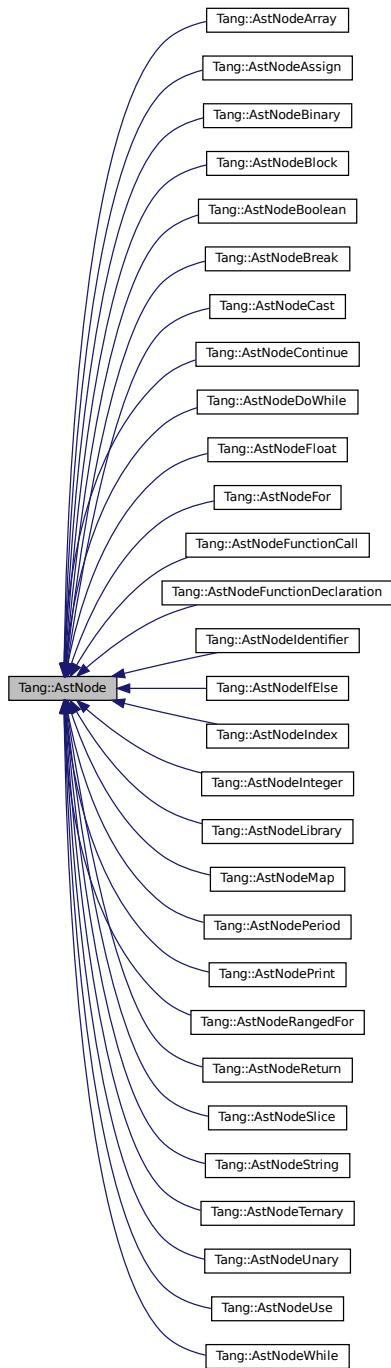
Class Documentation

5.1 Tang::AstNode Class Reference

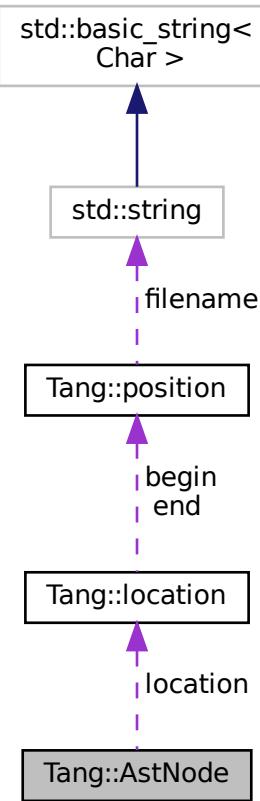
Base class for representing nodes of an Abstract Syntax Tree (AST).

```
#include <astNode.hpp>
```

Inheritance diagram for Tang::AstNode:



Collaboration diagram for Tang::AstNode:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNode (Tang::location location)`
The generic constructor.
- `virtual ~AstNode ()`
The object destructor.
- `virtual std::string dump (std::string indent="") const`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const`
Compile the ast of the provided Tang::Program.
- `virtual void compilePreprocess (Program &program, PreprocessState state) const`
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

5.1.1 Detailed Description

Base class for representing nodes of an Abstract Syntax Tree (AST).

By default, it will represent a NULL value. There will be *many* derived classes, each one conveying the syntactic meaning of the code that it represents.

5.1.2 Member Enumeration Documentation

5.1.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.1.3 Constructor & Destructor Documentation

5.1.3.1 AstNode()

```
AstNode::AstNode (   
    Tang::location location )
```

The generic constructor.

It should never be called on its own.

Parameters

<code>location</code>	The location associated with this node.
-----------------------	---

5.1.4 Member Function Documentation

5.1.4.1 compile()

```
void AstNode::compile (
    Tang::Program & program ) const [virtual]
```

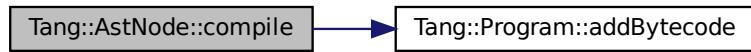
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUse](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#), [Tang::AstNodeMap](#), [Tang::AstNodeLibrary](#), [Tang::AstNodeInteger](#), [Tang::AstNodeIndex](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFunctionDeclaration](#), [Tang::AstNodeFunctionCall](#), [Tang::AstNodeFor](#), [Tang::AstNodeFloat](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeContinue](#), [Tang::AstNodeCast](#), [Tang::AstNodeBreak](#), [Tang::AstNodeBoolean](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), [Tang::AstNodeAssign](#), and [Tang::AstNodeArray](#).

Here is the call graph for this function:



5.1.4.2 compilePreprocess()

```
void AstNode::compilePreprocess (
    Program & program,
    PreprocessState state ) const [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUse](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#),

`Tang::AstNodeString`, `Tang::AstNodeSlice`, `Tang::AstNodeReturn`, `Tang::AstNodeRangedFor`, `Tang::AstNodePrint`, `Tang::AstNodePeriod`, `Tang::AstNodeMap`, `Tang::AstNodeLibrary`, `Tang::AstNodeIndex`, `Tang::AstNodeIfElse`, `Tang::AstNodeIdentifier`, `Tang::AstNodeFunctionDeclaration`, `Tang::AstNodeFunctionCall`, `Tang::AstNodeFor`, `Tang::AstNodeDoWhile`, `Tang::AstNodeCast`, `Tang::AstNodeBlock`, `Tang::AstNodeBinary`, `Tang::AstNodeAssign`, and `Tang::AstNodeArray`.

5.1.4.3 `dump()`

```
string AstNode::dump (
    std::string indent = "" ) const [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented in `Tang::AstNodeWhile`, `Tang::AstNodeUse`, `Tang::AstNodeUnary`, `Tang::AstNodeTernary`, `Tang::AstNodeString`, `Tang::AstNodeSlice`, `Tang::AstNodeReturn`, `Tang::AstNodeRangedFor`, `Tang::AstNodePrint`, `Tang::AstNodePeriod`, `Tang::AstNodeMap`, `Tang::AstNodeLibrary`, `Tang::AstNodeInteger`, `Tang::AstNodeIndex`, `Tang::AstNodeIfElse`, `Tang::AstNodeIdentifier`, `Tang::AstNodeFunctionDeclaration`, `Tang::AstNodeFunctionCall`, `Tang::AstNodeFor`, `Tang::AstNodeFloat`, `Tang::AstNodeDoWhile`, `Tang::AstNodeContinue`, `Tang::AstNodeCast`, `Tang::AstNodeBreak`, `Tang::AstNodeBoolean`, `Tang::AstNodeBlock`, `Tang::AstNodeBinary`, `Tang::AstNodeAssign`, and `Tang::AstNodeArray`.

Here is the call graph for this function:



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

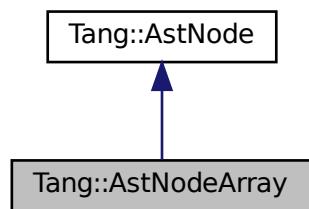
- `include/astNode.hpp`
- `src/astNode.cpp`

5.2 Tang::AstNodeArray Class Reference

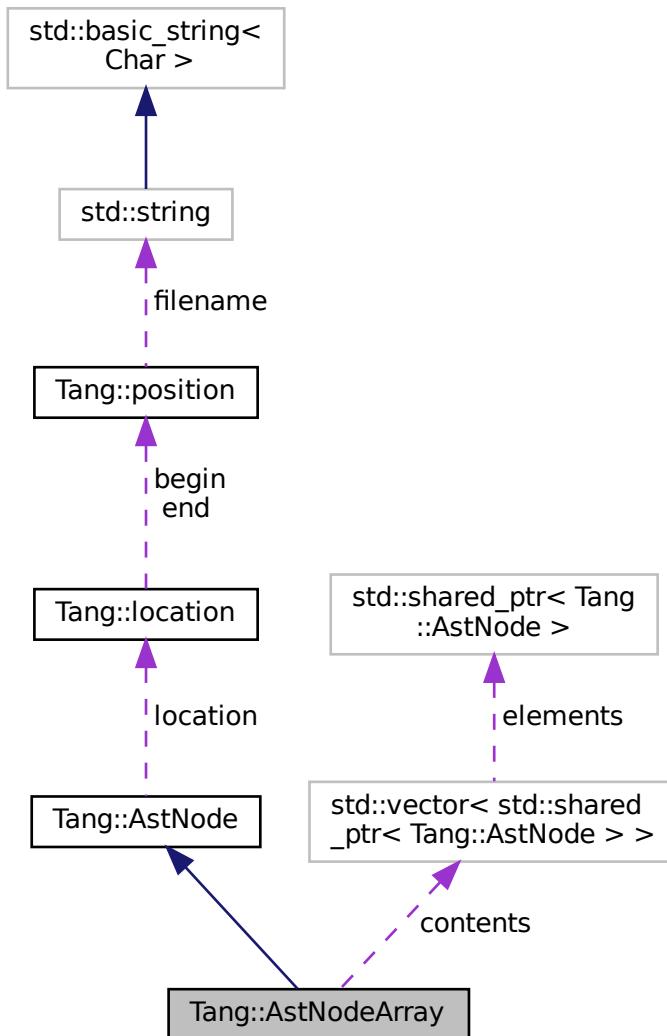
An [AstNode](#) that represents an array literal.

```
#include <astNodeArray.hpp>
```

Inheritance diagram for Tang::AstNodeArray:



Collaboration diagram for Tang::AstNodeArray:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeArray (std::vector< std::shared_ptr< Tang::AstNode >> contents, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`

Compile the ast of the provided `Tang::Program`.

- `virtual void compilePreprocess (Program &program, PreprocessState state) const override`
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `std::vector< std::shared_ptr< Tang::AstNode > > contents`
The contents of the array.

5.2.1 Detailed Description

An `AstNode` that represents an array literal.

5.2.2 Member Enumeration Documentation

5.2.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.2.3 Constructor & Destructor Documentation

5.2.3.1 AstNodeArray()

```
AstNodeArray::AstNodeArray (
    std::vector< std::shared_ptr< Tang::AstNode > >> contents,
    Tang::location location )
```

The constructor.

Parameters

<i>contents</i>	The contents of the array.
<i>location</i>	The location associated with the expression.

5.2.4 Member Function Documentation

5.2.4.1 compile()

```
void AstNodeArray::compile (
    Tang::Program & program ) const [override], [virtual]
```

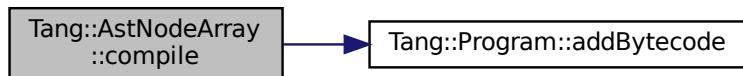
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.2.4.2 compilePreprocess()

```
void AstNodeArray::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.2.4.3 `dump()`

```
string AstNodeArray::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

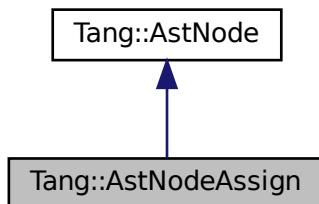
- [include/astNodeArray.hpp](#)
- [src/astNodeArray.cpp](#)

5.3 Tang::AstNodeAssign Class Reference

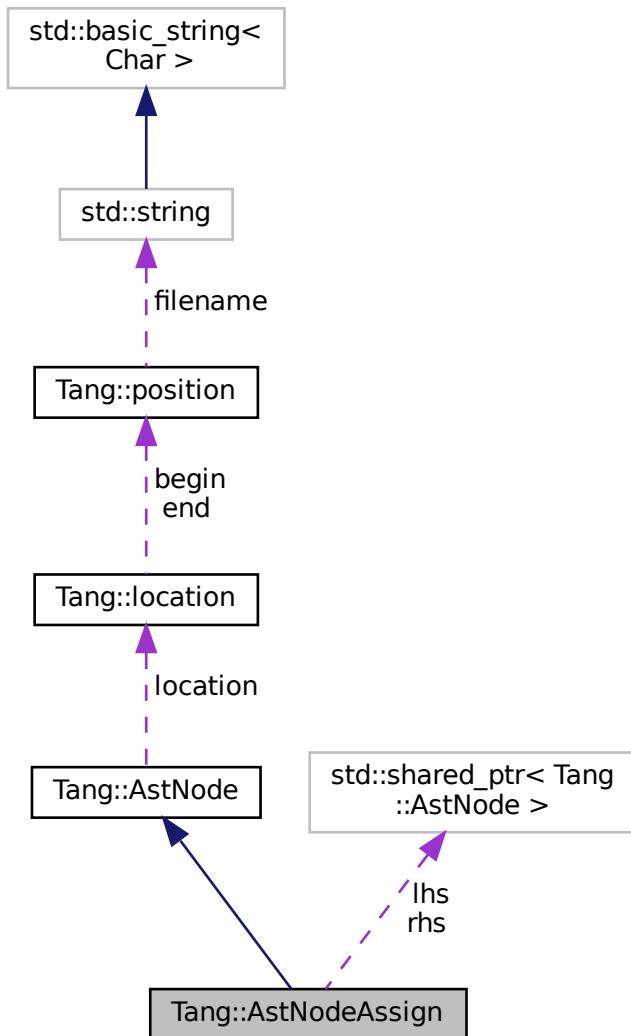
An [AstNode](#) that represents a binary expression.

```
#include <astNodeAssign.hpp>
```

Inheritance diagram for Tang::AstNodeAssign:



Collaboration diagram for Tang::AstNodeAssign:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeAssign (std::shared_ptr< AstNode > lhs, std::shared_ptr< AstNode > rhs, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`

Compile the ast of the provided [Tang::Program](#).

- `virtual void compilePreprocess (Program &program, PreprocessState state) const override`
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `std::shared_ptr< AstNode > lhs`
The left hand side expression.
- `std::shared_ptr< AstNode > rhs`
The right hand side expression.

5.3.1 Detailed Description

An [AstNode](#) that represents a binary expression.

5.3.2 Member Enumeration Documentation

5.3.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	AstNode is part of an assignment expression.

5.3.3 Constructor & Destructor Documentation

5.3.3.1 AstNodeAssign()

```
AstNodeAssign::AstNodeAssign (
    std::shared_ptr< AstNode > lhs,
```

```
std::shared_ptr< AstNode > rhs,
Tang::location location )
```

The constructor.

Parameters

<i>lhs</i>	The left hand side expression.
<i>rhs</i>	The right hand side expression.
<i>location</i>	The location associated with the expression.

5.3.4 Member Function Documentation

5.3.4.1 compile()

```
void AstNodeAssign::compile (
    Tang::Program & program ) const [override], [virtual]
```

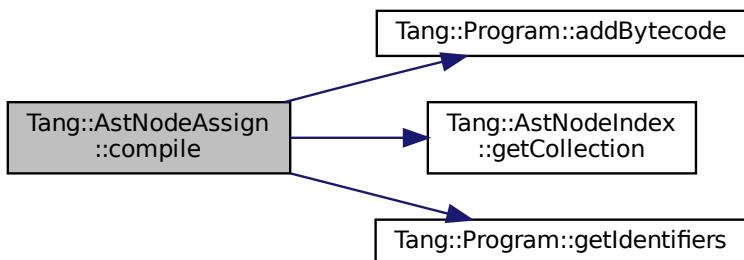
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.3.4.2 compilePreprocess()

```
void AstNodeAssign::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.3.4.3 dump()

```
string AstNodeAssign::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

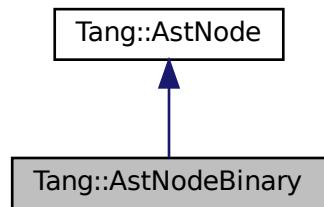
- [include/astNodeAssign.hpp](#)
- [src/astNodeAssign.cpp](#)

5.4 Tang::AstNodeBinary Class Reference

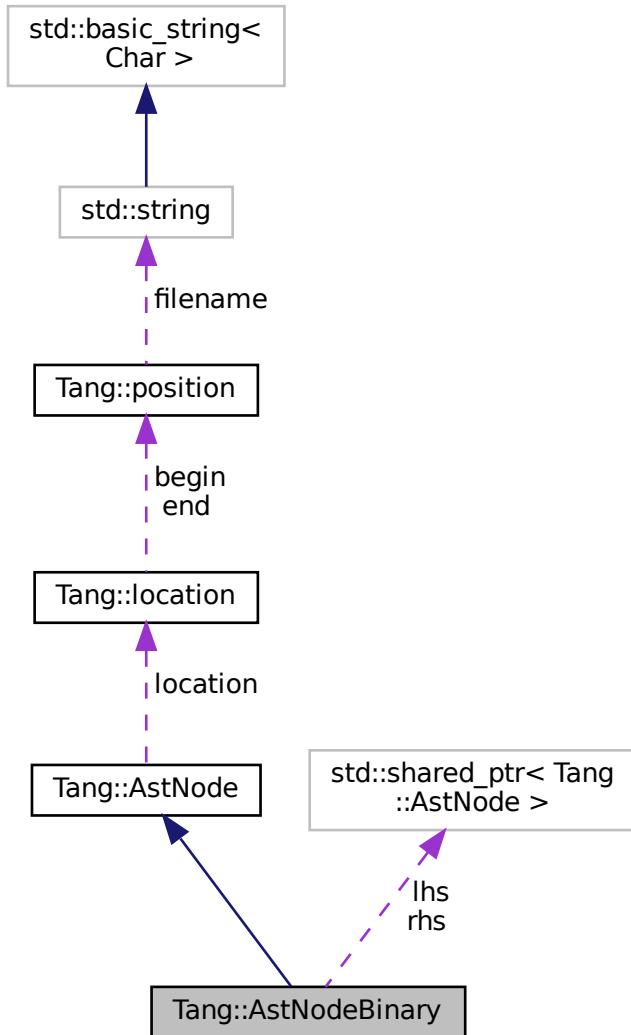
An [AstNode](#) that represents a binary expression.

```
#include <astNodeBinary.hpp>
```

Inheritance diagram for Tang::AstNodeBinary:



Collaboration diagram for Tang::AstNodeBinary:



Public Types

- enum **Operation** {
 Add , **Subtract** , **Multiply** , **Divide** ,
 Modulo , **LessThan** , **LessThanEqual** , **GreaterThan** ,
 GreaterThanEqual , **Equal** , **NotEqual** , **And** ,
 Or }

Indicates the type of binary expression that this node represents.
- enum **PreprocessState** : int { **Default** = 0 , **IsAssignment** = 1 }

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Public Member Functions

- `AstNodeBinary (Operation op, std::shared_ptr< AstNode > lhs, std::shared_ptr< AstNode > rhs, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`
Compile the ast of the provided Tang::Program.
- `virtual void compilePreprocess (Program &program, PreprocessState state) const override`
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `Operation op`
The binary operation performed.
- `std::shared_ptr< AstNode > lhs`
The left hand side expression.
- `std::shared_ptr< AstNode > rhs`
The right hand side expression.

5.4.1 Detailed Description

An `AstNode` that represents a binary expression.

5.4.2 Member Enumeration Documentation

5.4.2.1 Operation

```
enum Tang::AstNodeBinary::Operation
```

Indicates the type of binary expression that this node represents.

Enumerator

Add	Indicates lhs + rhs.
Subtract	Indicates lhs - rhs.
Multiply	Indicates lhs * rhs.
Divide	Indicates lhs / rhs.
Modulo	Indicates lhs % rhs.
LessThan	Indicates lhs < rhs.
LessThanEqual	Indicates lhs <= rhs.
GreaterThan	Indicates lhs > rhs.
GreaterThanEqual	Indicates lhs >= rhs.
Equal	Indicates lhs == rhs.

5.4.2.2 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	AstNode is part of an assignment expression.

5.4.3 Constructor & Destructor Documentation

5.4.3.1 AstNodeBinary()

```
AstNodeBinary::AstNodeBinary (
    Operation op,
    std::shared_ptr< AstNode > lhs,
    std::shared_ptr< AstNode > rhs,
    Tang::location location )
```

The constructor.

Parameters

<i>op</i>	The Tang::AstNodeBinary::Operation to perform.
<i>lhs</i>	The left hand side expression.
<i>rhs</i>	The right hand side expression.
<i>location</i>	The location associated with the expression.

5.4.4 Member Function Documentation

5.4.4.1 compile()

```
void AstNodeBinary::compile (
    Tang::Program & program ) const [override], [virtual]
```

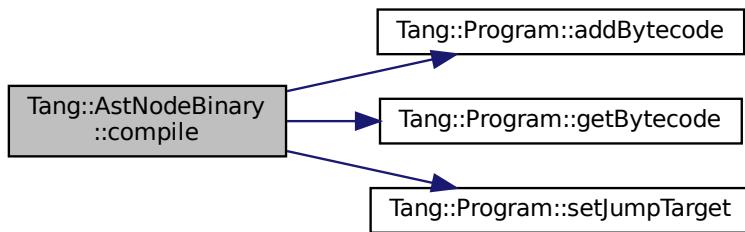
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Tang::Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.4.4.2 compilePreprocess()

```
void AstNodeBinary::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.4.4.3 dump()

```
string AstNodeBinary::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

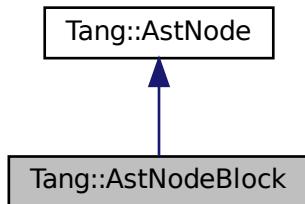
- [include/astNodeBinary.hpp](#)
- [src/astNodeBinary.cpp](#)

5.5 Tang::AstNodeBlock Class Reference

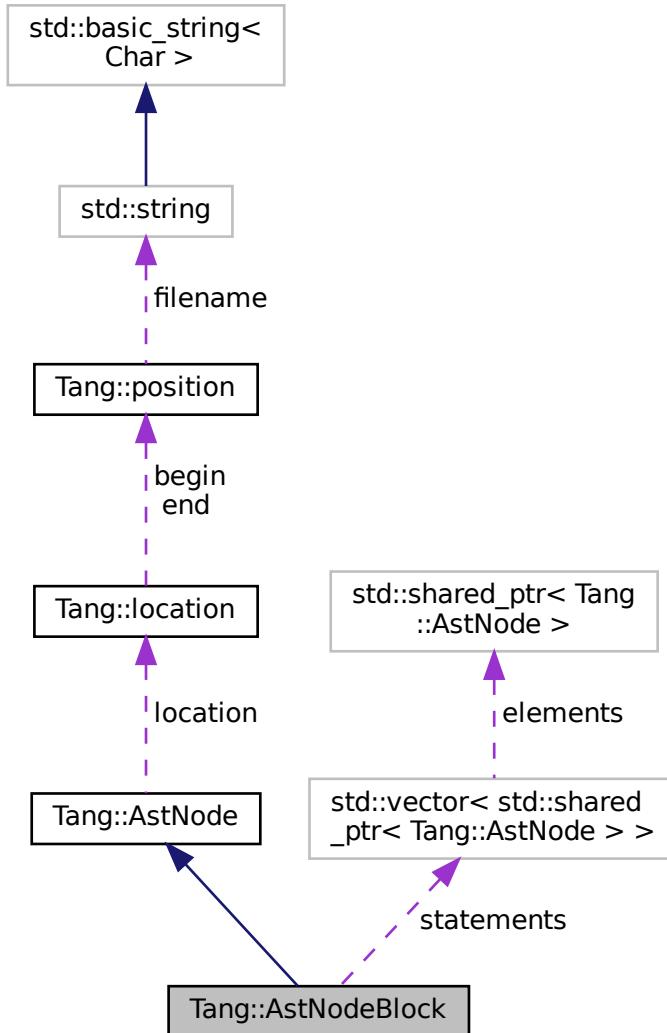
An [AstNode](#) that represents a code block.

```
#include <astNodeBlock.hpp>
```

Inheritance diagram for Tang::AstNodeBlock:



Collaboration diagram for Tang::AstNodeBlock:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeBlock` (const `std::vector<std::shared_ptr<AstNode>>` &`statements`, `Tang::location` `location`)
The constructor.
- virtual `std::string` `dump` (`std::string` `indent=""`) const override
Return a string that describes the contents of the node.
- virtual `void` `compile` (`Tang::Program` &`program`) const override

Compile the ast of the provided [Tang::Program](#).

- `virtual void compilePreprocess (Program &program, PreprocessState state) const override`
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `std::vector< std::shared_ptr< AstNode > > statements`
The statements included in the code block.

5.5.1 Detailed Description

An [AstNode](#) that represents a code block.

5.5.2 Member Enumeration Documentation

5.5.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	AstNode is part of an assignment expression.

5.5.3 Constructor & Destructor Documentation

5.5.3.1 AstNodeBlock()

```
AstNodeBlock::AstNodeBlock (
    const std::vector< std::shared_ptr< AstNode > >& statements,
    Tang::location location )
```

The constructor.

Parameters

<i>statements</i>	The statements of the code block.
<i>location</i>	The location associated with the expression.

5.5.4 Member Function Documentation

5.5.4.1 compile()

```
void AstNodeBlock::compile (
    Tang::Program & program ) const [override], [virtual]
```

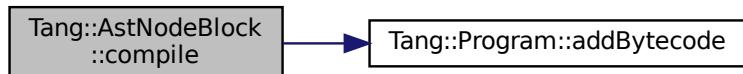
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.5.4.2 compilePreprocess()

```
void AstNodeBlock::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.5.4.3 `dump()`

```
string AstNodeBlock::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

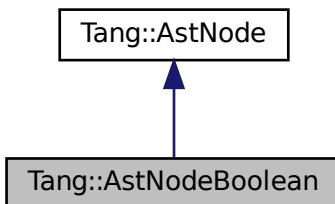
- [include/astNodeBlock.hpp](#)
- [src/astNodeBlock.cpp](#)

5.6 Tang::AstNodeBoolean Class Reference

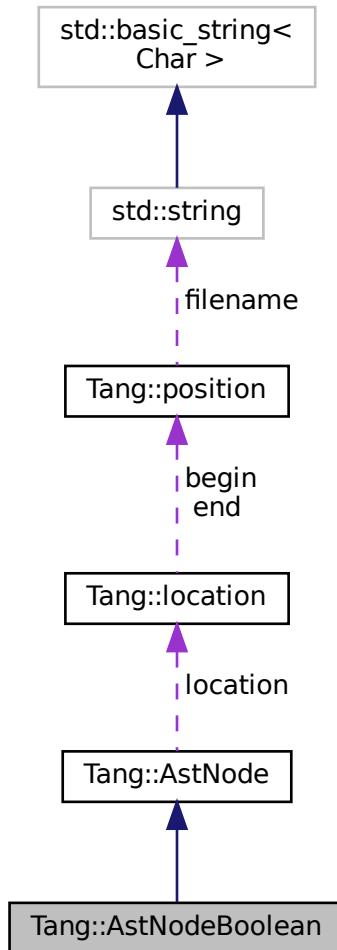
An [AstNode](#) that represents a boolean literal.

```
#include <astNodeBoolean.hpp>
```

Inheritance diagram for Tang::AstNodeBoolean:



Collaboration diagram for Tang::AstNodeBoolean:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeBoolean (bool val, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`
Compile the ast of the provided `Tang::Program`.
- `virtual void compilePreprocess (Program &program, PreprocessState state) const`
Run any preprocess analysis needed before compilation.

Protected Attributes

- [Tang::location location](#)
The location associated with this node.

Private Attributes

- [bool val](#)
The boolean value being stored.

5.6.1 Detailed Description

An [AstNode](#) that represents a boolean literal.

5.6.2 Member Enumeration Documentation

5.6.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	AstNode is part of an assignment expression.

5.6.3 Constructor & Destructor Documentation

5.6.3.1 AstNodeBoolean()

```
AstNodeBoolean::AstNodeBoolean (
    bool val,
    Tang::location location )
```

The constructor.

Parameters

<i>val</i>	The boolean to represent.
<i>location</i>	The location associated with the expression.

5.6.4 Member Function Documentation

5.6.4.1 compile()

```
void AstNodeBoolean::compile (
    Tang::Program & program ) const [override], [virtual]
```

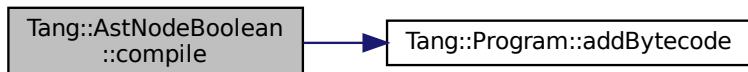
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.6.4.2 compilePreprocess()

```
void AstNode::compilePreprocess (
    Program & program,
    PreprocessState state ) const [virtual], [inherited]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUse](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#), [Tang::AstNodeMap](#), [Tang::AstNodeLibrary](#), [Tang::AstNodeIndex](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFunctionDeclaration](#), [Tang::AstNodeFunctionCall](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeCast](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), [Tang::AstNodeAssign](#), and [Tang::AstNodeArray](#).

5.6.4.3 `dump()`

```
string AstNodeBoolean::dump (
```

	<code>std::string indent = ""</code>	<code>) const [override], [virtual]</code>
--	--------------------------------------	--

Return a string that describes the contents of the node.

Parameters

<code>indent</code>	A string used to indent the dump.
---------------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

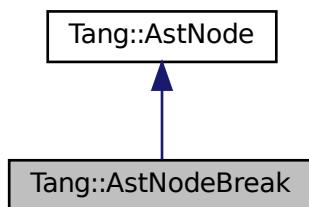
- [include/astNodeBoolean.hpp](#)
- [src/astNodeBoolean.cpp](#)

5.7 Tang::AstNodeBreak Class Reference

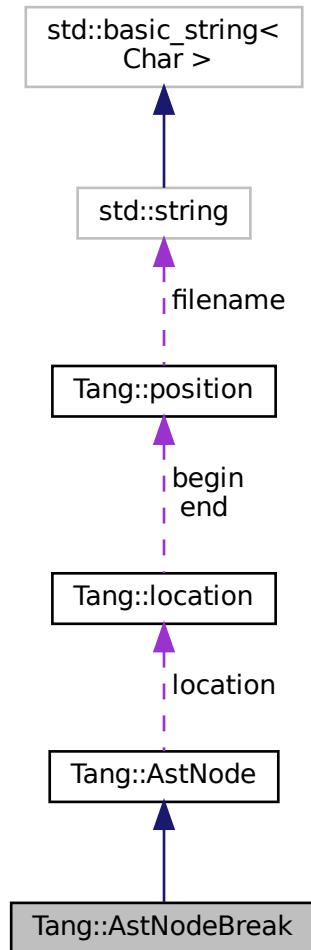
An [AstNode](#) that represents a `break` statement.

```
#include <astNodeBreak.hpp>
```

Inheritance diagram for Tang::AstNodeBreak:



Collaboration diagram for Tang::AstNodeBreak:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeBreak (Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`
Compile the ast of the provided `Tang::Program`.
- `virtual void compilePreprocess (Program &program, PreprocessState state) const`
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

5.7.1 Detailed Description

An `AstNode` that represents a `break` statement.

5.7.2 Member Enumeration Documentation

5.7.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.7.3 Constructor & Destructor Documentation

5.7.3.1 AstNodeBreak()

```
AstNodeBreak::AstNodeBreak ( Tang::location location )
```

The constructor.

Parameters

<code>location</code>	The location associated with the expression.
-----------------------	--

5.7.4 Member Function Documentation

5.7.4.1 compile()

```
void AstNodeBreak::compile (
    Tang::Program & program ) const [override], [virtual]
```

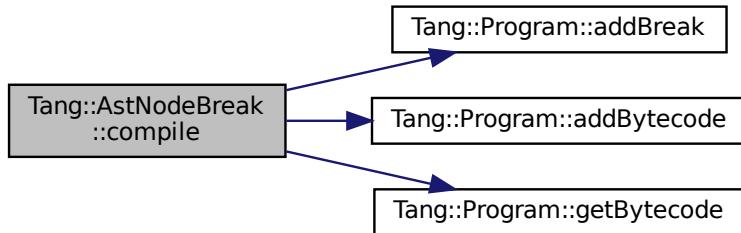
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.7.4.2 compilePreprocess()

```
void AstNode::compilePreprocess (
    Program & program,
    PreprocessState state ) const [virtual], [inherited]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUse](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#), [Tang::AstNodeMap](#), [Tang::AstNodeLibrary](#), [Tang::AstNodeIndex](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFunctionDeclaration](#), [Tang::AstNodeFunctionCall](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeCast](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), [Tang::AstNodeAssign](#), and [Tang::AstNodeArray](#).

5.7.4.3 `dump()`

```
string AstNodeBreak::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

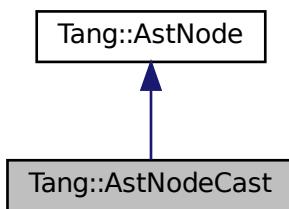
- [include/astNodeBreak.hpp](#)
- [src/astNodeBreak.cpp](#)

5.8 Tang::AstNodeCast Class Reference

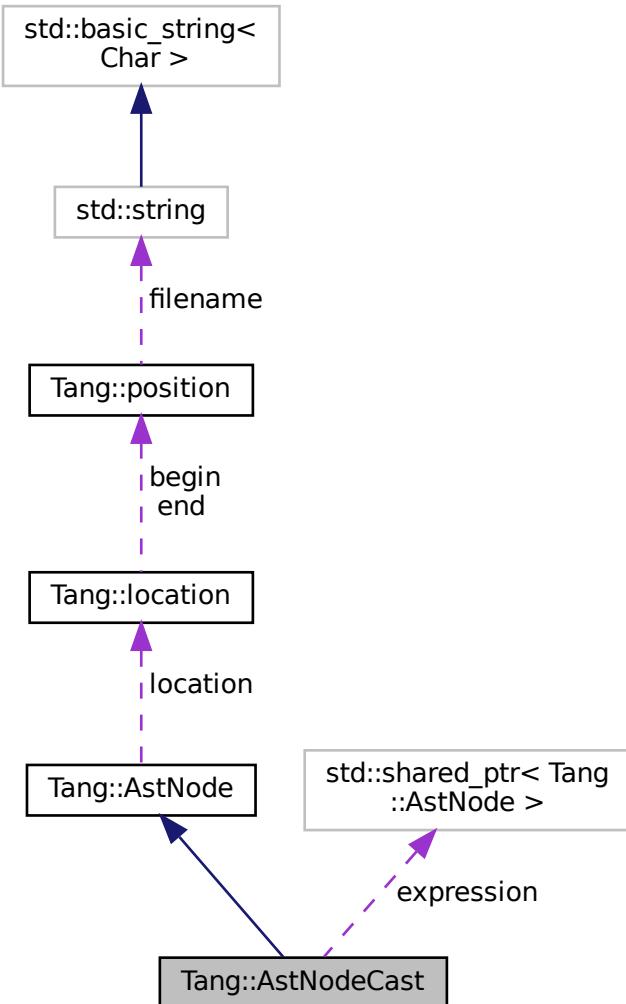
An [AstNode](#) that represents a typecast of an expression.

```
#include <astNodeCast.hpp>
```

Inheritance diagram for Tang::AstNodeCast:



Collaboration diagram for Tang::AstNodeCast:



Public Types

- enum `Type` { `Integer` , `Float` , `Boolean` , `String` }
The possible types that can be cast to.
- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Public Member Functions

- `AstNodeCast (Type targetType, std::shared_ptr<AstNode> expression, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`

Return a string that describes the contents of the node.

- virtual void `compile (Tang::Program &program)` const override
Compile the ast of the provided Tang::Program.
- virtual void `compilePreprocess (Program &program, PreprocessState state)` const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `Type targetType`
The target type.
- `std::shared_ptr< AstNode > expression`
The expression being typecast.

5.8.1 Detailed Description

An `AstNode` that represents a typecast of an expression.

5.8.2 Member Enumeration Documentation

5.8.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.8.2.2 Type

```
enum Tang::AstNodeCast::Type
```

The possible types that can be cast to.

Enumerator

Integer	Cast to a Tang::ComputedExpressionInteger .
Float	Cast to a Tang::ComputedExpressionFloat .
Boolean	Cast to a Tang::ComputedExpressionBoolean .
String	Cast to a Tang::ComputedExpressionString .

5.8.3 Constructor & Destructor Documentation**5.8.3.1 AstNodeCast()**

```
AstNodeCast::AstNodeCast (
    Type targetType,
    std::shared_ptr< AstNode > expression,
    Tang::location location )
```

The constructor.

Parameters

<i>targetType</i>	The target type that the expression will be cast to.
<i>expression</i>	The expression to be typecast.
<i>location</i>	The location associated with this node.

5.8.4 Member Function Documentation**5.8.4.1 compile()**

```
void AstNodeCast::compile (
    Tang::Program & program ) const [override], [virtual]
```

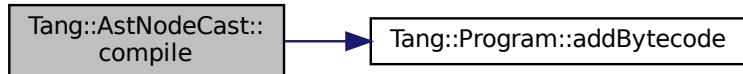
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.8.4.2 compilePreprocess()

```
void AstNodeCast::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.8.4.3 dump()

```
string AstNodeCast::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

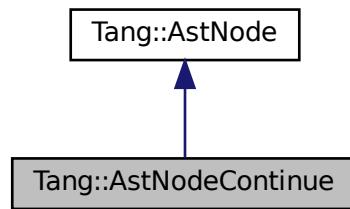
- [include/astNodeCast.hpp](#)
- [src/astNodeCast.cpp](#)

5.9 Tang::AstNodeContinue Class Reference

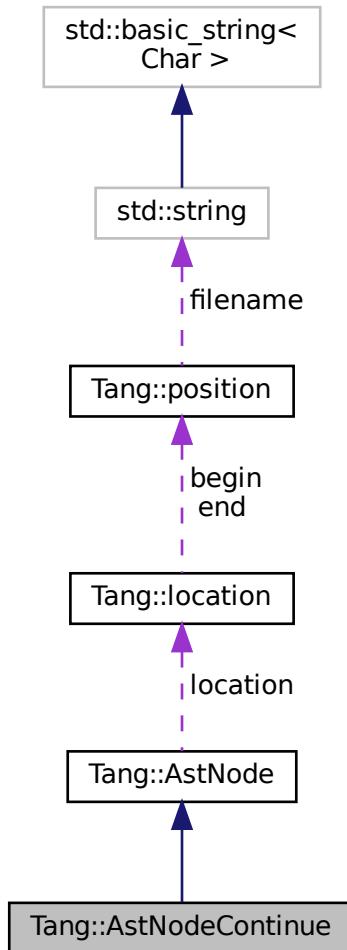
An [AstNode](#) that represents a `continue` statement.

```
#include <astNodeContinue.hpp>
```

Inheritance diagram for Tang::AstNodeContinue:



Collaboration diagram for Tang::AstNodeContinue:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeContinue (Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`
Compile the ast of the provided `Tang::Program`.
- `virtual void compilePreprocess (Program &program, PreprocessState state) const`
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

5.9.1 Detailed Description

An `AstNode` that represents a `continue` statement.

5.9.2 Member Enumeration Documentation

5.9.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.9.3 Constructor & Destructor Documentation

5.9.3.1 AstNodeContinue()

```
AstNodeContinue::AstNodeContinue (
    Tang::location location )
```

The constructor.

Parameters

<code>location</code>	The location associated with the expression.
-----------------------	--

5.9.4 Member Function Documentation

5.9.4.1 compile()

```
void AstNodeContinue::compile (
    Tang::Program & program ) const [override], [virtual]
```

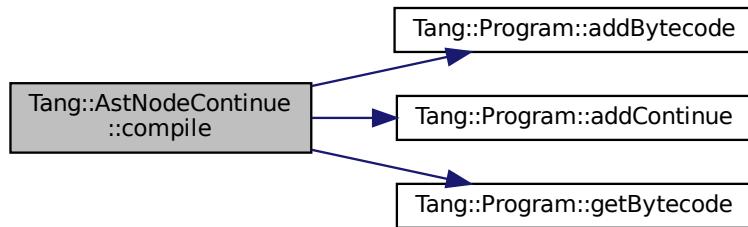
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.9.4.2 compilePreprocess()

```
void AstNode::compilePreprocess (
    Program & program,
    PreprocessState state ) const [virtual], [inherited]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUse](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#), [Tang::AstNodeMap](#), [Tang::AstNodeLibrary](#), [Tang::AstNodeIndex](#), [Tang::AstNodeElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFunctionDeclaration](#), [Tang::AstNodeFunctionCall](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeCast](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), [Tang::AstNodeAssign](#), and [Tang::AstNodeArray](#).

5.9.4.3 `dump()`

```
string AstNodeContinue::dump (
    std::string indent = "") const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

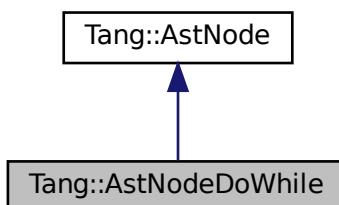
- [include/astNodeContinue.hpp](#)
- [src/astNodeContinue.cpp](#)

5.10 `Tang::AstNodeDoWhile` Class Reference

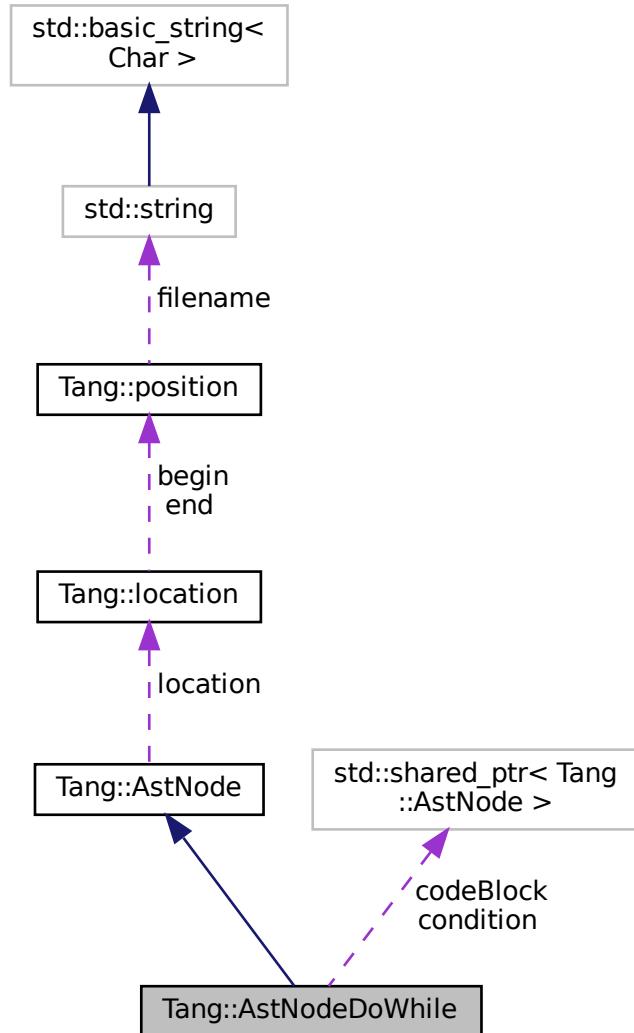
An [AstNode](#) that represents a do..while statement.

```
#include <astNodeDoWhile.hpp>
```

Inheritance diagram for `Tang::AstNodeDoWhile`:



Collaboration diagram for Tang::AstNodeDoWhile:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeDoWhile (std::shared_ptr< AstNode > condition, std::shared_ptr< AstNode > codeBlock, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.

- virtual void `compile (Tang::Program &program)` const override
Compile the ast of the provided `Tang::Program`.
- virtual void `compilePreprocess (Program &program, PreprocessState state)` const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `std::shared_ptr< AstNode > condition`
The expression which determines whether or not the code block will continue to be executed.
- `std::shared_ptr< AstNode > codeBlock`
The code block executed when the condition is true.

5.10.1 Detailed Description

An `AstNode` that represents a do..while statement.

5.10.2 Member Enumeration Documentation

5.10.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.10.3 Constructor & Destructor Documentation

5.10.3.1 AstNodeDoWhile()

```
AstNodeDoWhile::AstNodeDoWhile (
    std::shared_ptr< AstNode > condition,
```

```
std::shared_ptr< AstNode > codeBlock,
Tang::location location )
```

The constructor.

Parameters

<i>condition</i>	The expression which determines whether the thenBlock or elseBlock is executed.
<i>codeBlock</i>	The statement executed when the condition is true.
<i>location</i>	The location associated with the expression.

5.10.4 Member Function Documentation

5.10.4.1 compile()

```
void AstNodeDoWhile::compile (
    Tang::Program & program ) const [override], [virtual]
```

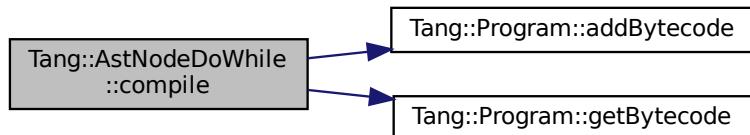
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.10.4.2 compilePreprocess()

```
void AstNodeDoWhile::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.10.4.3 `dump()`

```
string AstNodeDoWhile::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

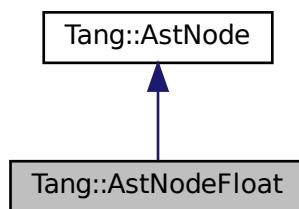
- `include/astNodeDoWhile.hpp`
- `src/astNodeDoWhile.cpp`

5.11 Tang::AstNodeFloat Class Reference

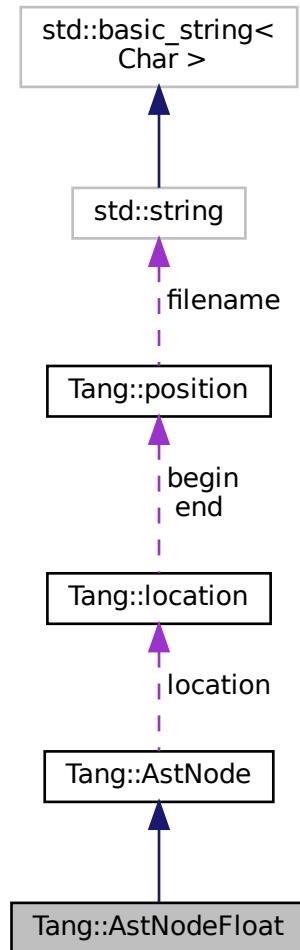
An [AstNode](#) that represents an float literal.

```
#include <astNodeFloat.hpp>
```

Inheritance diagram for Tang::AstNodeFloat:



Collaboration diagram for Tang::AstNodeFloat:



Public Types

- enum **PreprocessState** : int { **Default** = 0 , **IsAssignment** = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- AstNodeFloat (Tang::float_t number, Tang::location location)**
The constructor.
- virtual std::string dump (std::string indent="") const override**
Return a string that describes the contents of the node.
- virtual void compile (Tang::Program &program) const override**
Compile the ast of the provided Tang::Program.
- virtual void compilePreprocess (Program &program, PreprocessState state) const**
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `Tang::float_t val`
The float value being stored.

5.11.1 Detailed Description

An `AstNode` that represents an float literal.

Integers are represented by the `Tang::float_t` type, and so are limited in range by that of the underlying type.

5.11.2 Member Enumeration Documentation

5.11.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.11.3 Constructor & Destructor Documentation

5.11.3.1 AstNodeFloat()

```
AstNodeFloat::AstNodeFloat (
    Tang::float_t number,
    Tang::location location )
```

The constructor.

Parameters

<i>number</i>	The number to represent.
<i>location</i>	The location associated with the expression.

5.11.4 Member Function Documentation**5.11.4.1 compile()**

```
void AstNodeFloat::compile (
    Tang::Program & program ) const [override], [virtual]
```

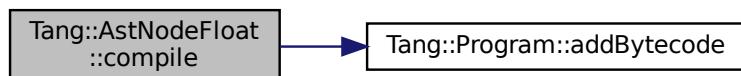
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:

**5.11.4.2 compilePreprocess()**

```
void AstNode::compilePreprocess (
    Program & program,
    PreprocessState state ) const [virtual], [inherited]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUse](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#), [Tang::AstNodeMap](#), [Tang::AstNodeLibrary](#), [Tang::AstNodeIndex](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFunctionDeclaration](#), [Tang::AstNodeFunctionCall](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeCast](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), [Tang::AstNodeAssign](#), and [Tang::AstNodeArray](#).

5.11.4.3 `dump()`

```
string AstNodeFloat::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

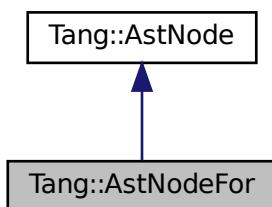
- [include/astNodeFloat.hpp](#)
- [src/astNodeFloat.cpp](#)

5.12 Tang::AstNodeFor Class Reference

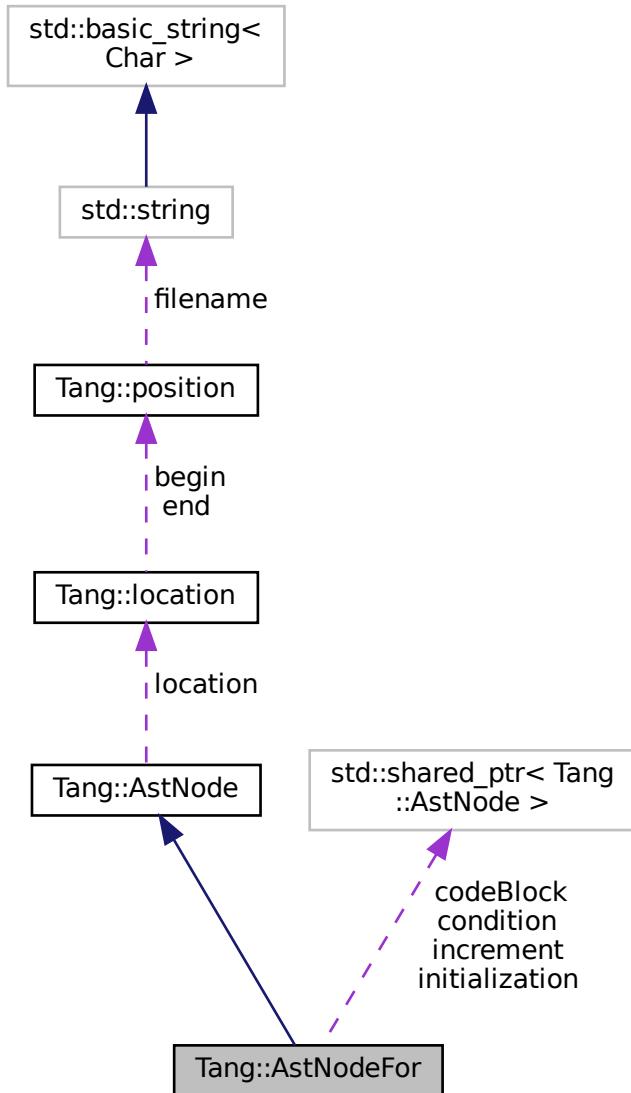
An [AstNode](#) that represents an if() statement.

```
#include <astNodeFor.hpp>
```

Inheritance diagram for Tang::AstNodeFor:



Collaboration diagram for Tang::AstNodeFor:



Public Types

- enum **PreprocessState** : int { **Default** = 0 , **IsAssignment** = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- AstNodeFor** (`std::shared_ptr<AstNode> initialization, std::shared_ptr<AstNode> condition, std::shared_ptr<AstNode> increment, std::shared_ptr<AstNode> codeBlock, Tang::location location)`)
The constructor.

- virtual std::string `dump` (std::string `indent=""`) const override
Return a string that describes the contents of the node.
- virtual void `compile` (`Tang::Program` &`program`) const override
Compile the ast of the provided `Tang::Program`.
- virtual void `compilePreprocess` (`Program` &`program`, `PreprocessState` `state`) const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location` `location`
The location associated with this node.

Private Attributes

- `std::shared_ptr< AstNode >` `initialization`
The expression to be executed first to set up the for() loop.
- `std::shared_ptr< AstNode >` `condition`
The expression which determines whether or not the code block will continue to be executed.
- `std::shared_ptr< AstNode >` `increment`
The expression to be executed immediately after the code block.
- `std::shared_ptr< AstNode >` `codeBlock`
The code block executed when the condition is true.

5.12.1 Detailed Description

An `AstNode` that represents an if() statement.

5.12.2 Member Enumeration Documentation

5.12.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

<code>Default</code>	The default state.
<code>IsAssignment</code>	<code>AstNode</code> is part of an assignment expression.

5.12.3 Constructor & Destructor Documentation

5.12.3.1 AstNodeFor()

```
AstNodeFor::AstNodeFor (
    std::shared_ptr< AstNode > initialization,
    std::shared_ptr< AstNode > condition,
    std::shared_ptr< AstNode > increment,
    std::shared_ptr< AstNode > codeBlock,
    Tang::location location )
```

The constructor.

Parameters

<i>initialization</i>	The expression to be executed first.
<i>condition</i>	The expression which determines whether the codeBlock is executed.
<i>increment</i>	The expression to be executed after each codeBlock.
<i>codeBlock</i>	The statement executed when the condition is true.
<i>location</i>	The location associated with the expression.

5.12.4 Member Function Documentation

5.12.4.1 compile()

```
void AstNodeFor::compile (
    Tang::Program & program ) const [override], [virtual]
```

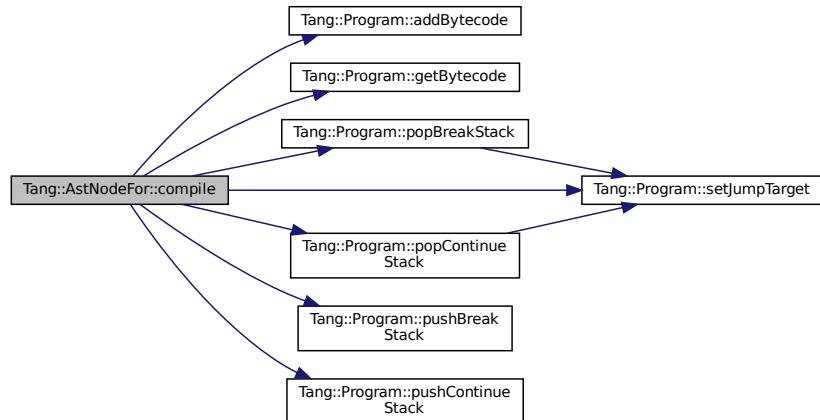
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.12.4.2 compilePreprocess()

```
void AstNodeFor::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.12.4.3 dump()

```
string AstNodeFor::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

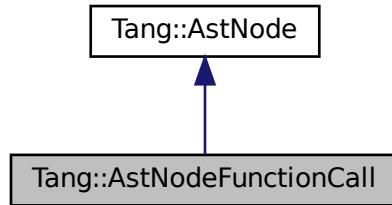
- [include/astNodeFor.hpp](#)
- [src/astNodeFor.cpp](#)

5.13 Tang::AstNodeFunctionCall Class Reference

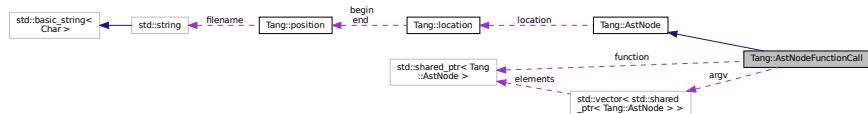
An [AstNode](#) that represents a function call.

```
#include <astNodeFunctionCall.hpp>
```

Inheritance diagram for Tang::AstNodeFunctionCall:



Collaboration diagram for Tang::AstNodeFunctionCall:



Public Types

- enum [PreprocessState](#) : int { [Default](#) = 0 , [IsAssignment](#) = 1 }

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Public Member Functions

- `AstNodeFunctionCall` (`std::shared_ptr< AstNode > function, std::vector< std::shared_ptr< AstNode > > argv, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`
Compile the ast of the provided Tang::Program.
- `virtual void compilePreprocess (Program &program, PreprocessState state) const override`
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `std::shared_ptr< AstNode > function`
The function being invoked.
- `std::vector< std::shared_ptr< AstNode > > argv`
The list of arguments provided to the function.

5.13.1 Detailed Description

An `AstNode` that represents a function call.

5.13.2 Member Enumeration Documentation

5.13.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.13.3 Constructor & Destructor Documentation

5.13.3.1 AstNodeFunctionCall()

```
AstNodeFunctionCall::AstNodeFunctionCall (
    std::shared_ptr< AstNode > function,
    std::vector< std::shared_ptr< AstNode > >> argv,
    Tang::location location )
```

The constructor.

Parameters

<i>function</i>	The function being invoked.
<i>argv</i>	The list of arguments provided to the function.
<i>location</i>	The location associated with the expression.

5.13.4 Member Function Documentation

5.13.4.1 compile()

```
void AstNodeFunctionCall::compile (
    Tang::Program & program ) const [override], [virtual]
```

Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.13.4.2 compilePreprocess()

```
void AstNodeFunctionCall::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.13.4.3 dump()

```
string AstNodeFunctionCall::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

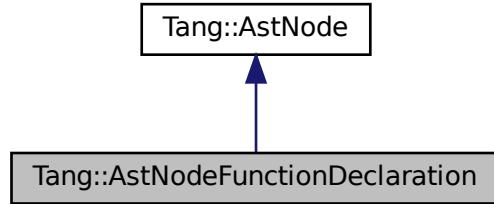
- [include/astNodeFunctionCall.hpp](#)
- [src/astNodeFunctionCall.cpp](#)

5.14 Tang::AstNodeFunctionDeclaration Class Reference

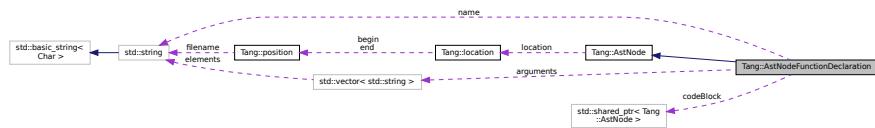
An [AstNode](#) that represents a function declaration.

```
#include <astNodeFunctionDeclaration.hpp>
```

Inheritance diagram for Tang::AstNodeFunctionDeclaration:



Collaboration diagram for Tang::AstNodeFunctionDeclaration:



Public Types

- enum [PreprocessState](#) : int { [Default](#) = 0 , [IsAssignment](#) = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- [AstNodeFunctionDeclaration](#) (std::string [name](#), std::vector< std::string > [arguments](#), std::shared_ptr< [AstNode](#) > [codeBlock](#), [Tang::location](#) [location](#))
The constructor.
- virtual std::string [dump](#) (std::string [indent](#)= "") const override
Return a string that describes the contents of the node.
- virtual void [compile](#) ([Tang::Program](#) &[program](#)) const override
Compile the ast of the provided [Tang::Program](#).
- virtual void [compilePreprocess](#) ([Program](#) &[program](#), [PreprocessState](#) [state](#)) const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- [Tang::location](#) [location](#)
The location associated with this node.

Private Attributes

- std::string `name`
The name of the function.
- std::vector< std::string > `arguments`
The arguments expected to be provided.
- std::shared_ptr< `AstNode` > `codeBlock`
The code block executed when the condition is true.

5.14.1 Detailed Description

An `AstNode` that represents a function declaration.

5.14.2 Member Enumeration Documentation

5.14.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.14.3 Constructor & Destructor Documentation

5.14.3.1 AstNodeFunctionDeclaration()

```
AstNodeFunctionDeclaration::AstNodeFunctionDeclaration (
    std::string name,
    std::vector< std::string > arguments,
    std::shared_ptr< AstNode > codeBlock,
    Tang::location location )
```

The constructor.

Parameters

<code>name</code>	The name of the function.
<code>arguments</code>	The arguments expected to be provided.
<code>codeBlock</code>	The code executed as part of the function.
<code>location</code>	The location associated with the function declaration.

5.14.4 Member Function Documentation

5.14.4.1 compile()

```
void AstNodeFunctionDeclaration::compile (
    Tang::Program & program ) const [override], [virtual]
```

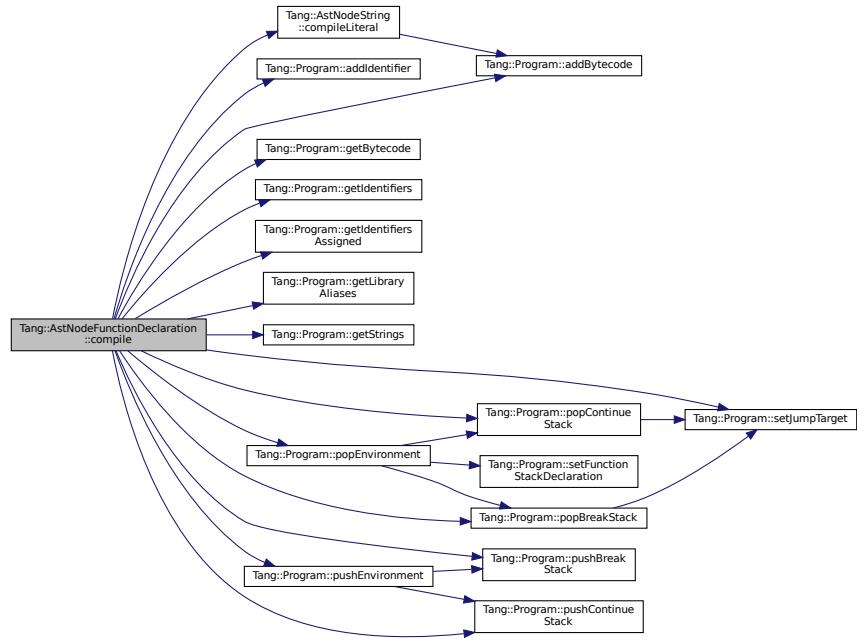
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.14.4.2 compilePreprocess()

```
void AstNodeFunctionDeclaration::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.14.4.3 `dump()`

```
string AstNodeFunctionDeclaration::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

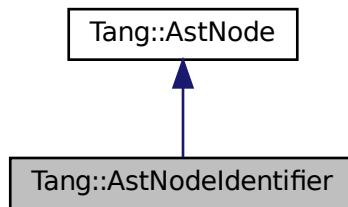
- [include/astNodeFunctionDeclaration.hpp](#)
- [src/astNodeFunctionDeclaration.cpp](#)

5.15 Tang::AstNodeIdentifier Class Reference

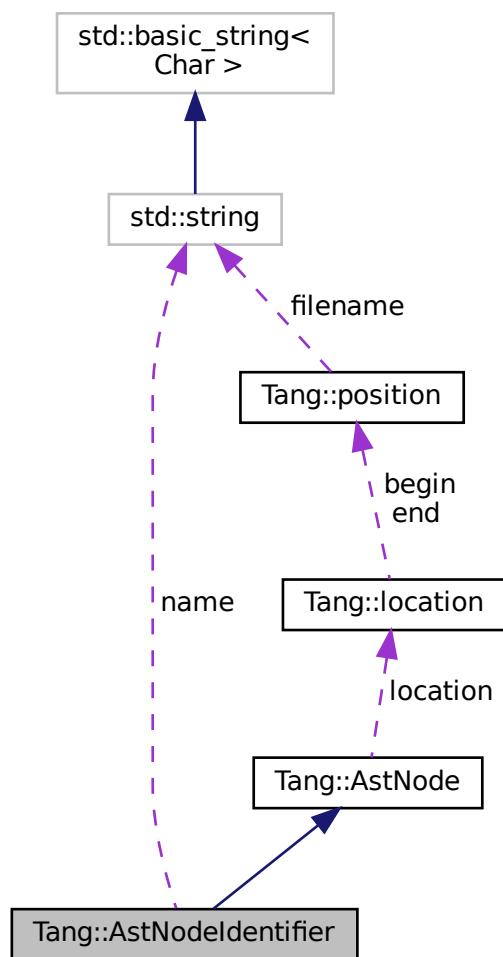
An [AstNode](#) that represents an identifier.

```
#include <astNodeIdentifier.hpp>
```

Inheritance diagram for Tang::AstNodeIdentifier:



Collaboration diagram for Tang::AstNodeIdentifier:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeIdentifier` (const std::string &`name`, `Tang::location` `location`)
The constructor.
- virtual std::string `dump` (std::string `indent`= "") const override
Return a string that describes the contents of the node.
- virtual void `compile` (`Tang::Program` &`program`) const override
Compile the ast of the provided `Tang::Program`.
- virtual void `compilePreprocess` (`Program` &`program`, `PreprocessState` `state`) const override
Run any preprocess analysis needed before compilation.

Public Attributes

- std::string `name`
The name of the identifier.

Protected Attributes

- `Tang::location` `location`
The location associated with this node.

5.15.1 Detailed Description

An `AstNode` that represents an identifier.

Identifier names are represented by a string.

5.15.2 Member Enumeration Documentation

5.15.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

<code>Default</code>	The default state.
<code>IsAssignment</code>	<code>AstNode</code> is part of an assignment expression.

5.15.3 Constructor & Destructor Documentation

5.15.3.1 AstNodeIdentifier()

```
AstNodeIdentifier::AstNodeIdentifier (
    const std::string & name,
    Tang::location location )
```

The constructor.

Parameters

<i>name</i>	The name of the identifier
<i>location</i>	The location associated with the expression.

5.15.4 Member Function Documentation

5.15.4.1 compile()

```
void AstNodeIdentifier::compile (
    Tang::Program & program ) const [override], [virtual]
```

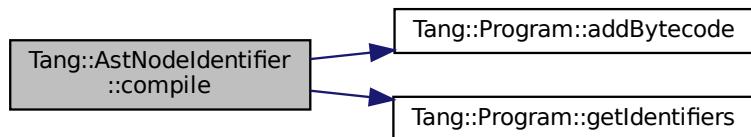
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.15.4.2 compilePreprocess()

```
void AstNodeIdentifier::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

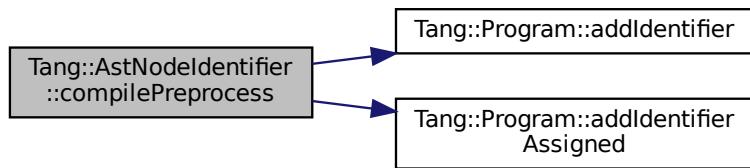
Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.15.4.3 dump()

```
string AstNodeIdentifier::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

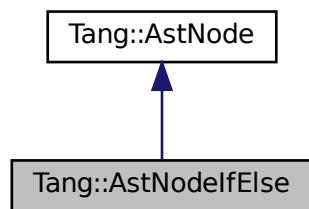
- [include/astNodeIdentifier.hpp](#)
- [src/astNodeIdentifier.cpp](#)

5.16 Tang::AstNodeIfElse Class Reference

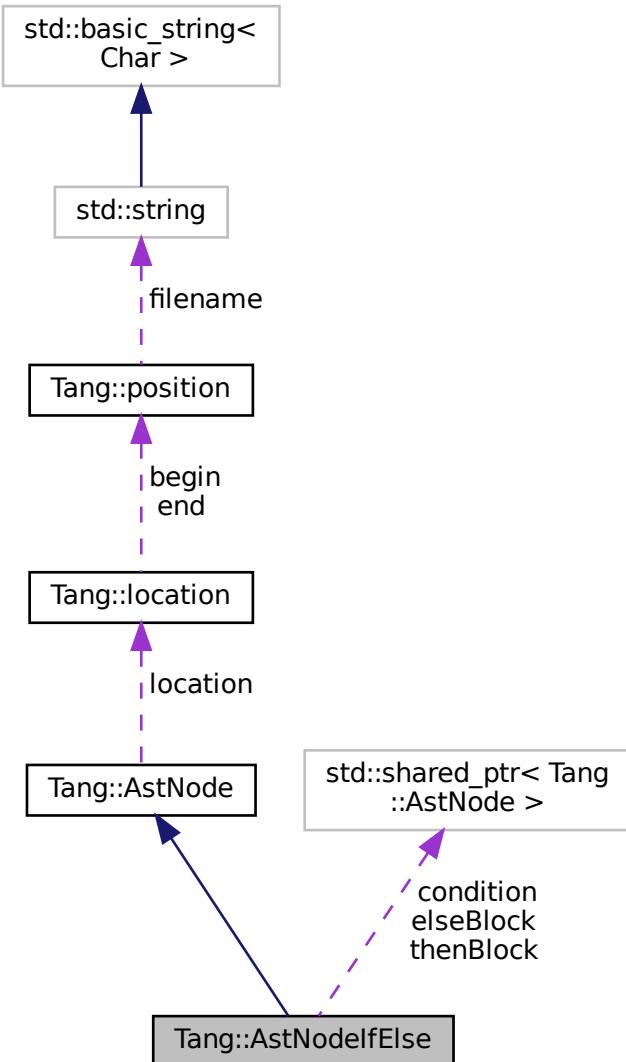
An [AstNode](#) that represents an if..else statement.

```
#include <astNodeIfElse.hpp>
```

Inheritance diagram for Tang::AstNodeIfElse:



Collaboration diagram for Tang::AstNodeIfElse:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeIfElse` (`std::shared_ptr< AstNode > condition`, `std::shared_ptr< AstNode > thenBlock`, `std::shared_ptr< AstNode > elseBlock`, `Tang::location location`)
- The constructor.*

- `AstNodeIfElse` (`std::shared_ptr< AstNode >` `condition`, `std::shared_ptr< AstNode >` `thenBlock`, `Tang::location location`)
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`
Compile the ast of the provided `Tang::Program`.
- `virtual void compilePreprocess (Program &program, PreprocessState state) const override`
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `std::shared_ptr< AstNode > condition`
The expression which determines whether the `thenBlock` or `elseBlock` is executed.
- `std::shared_ptr< AstNode > thenBlock`
The statement executed when the condition is true.
- `std::shared_ptr< AstNode > elseBlock`
The statement executed when the condition is false.

5.16.1 Detailed Description

An `AstNode` that represents an if..else statement.

5.16.2 Member Enumeration Documentation

5.16.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.16.3 Constructor & Destructor Documentation

5.16.3.1 `AstNodeIfElse()` [1/2]

```
AstNodeIfElse::AstNodeIfElse (
    std::shared_ptr< AstNode > condition,
    std::shared_ptr< AstNode > thenBlock,
    std::shared_ptr< AstNode > elseBlock,
    Tang::location location )
```

The constructor.

Parameters

<i>condition</i>	The expression which determines whether the thenBlock or elseBlock is executed.
<i>thenBlock</i>	The statement executed when the condition is true.
<i>elseBlock</i>	The statement executed when the condition is false.
<i>location</i>	The location associated with the expression.

5.16.3.2 `AstNodeIfElse()` [2/2]

```
AstNodeIfElse::AstNodeIfElse (
    std::shared_ptr< AstNode > condition,
    std::shared_ptr< AstNode > thenBlock,
    Tang::location location )
```

The constructor.

Parameters

<i>condition</i>	The expression which determines whether the thenBlock or elseBlock is executed.
<i>thenBlock</i>	The statement executed when the condition is true.
<i>location</i>	The location associated with the expression.

5.16.4 Member Function Documentation

5.16.4.1 `compile()`

```
void AstNodeIfElse::compile (
    Tang::Program & program ) const [override], [virtual]
```

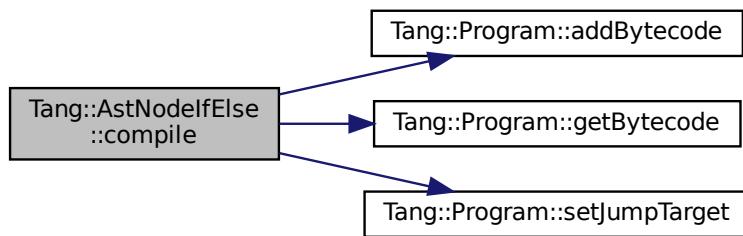
Compile the ast of the provided `Tang::Program`.

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.16.4.2 compilePreprocess()

```
void AstNodeIfElse::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.16.4.3 dump()

```
string AstNodeIfElse::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

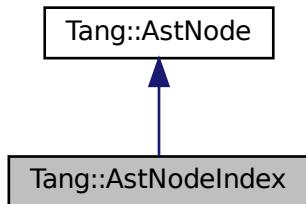
- [include/astNodeIfElse.hpp](#)
- [src/astNodeIfElse.cpp](#)

5.17 Tang::AstNodeIndex Class Reference

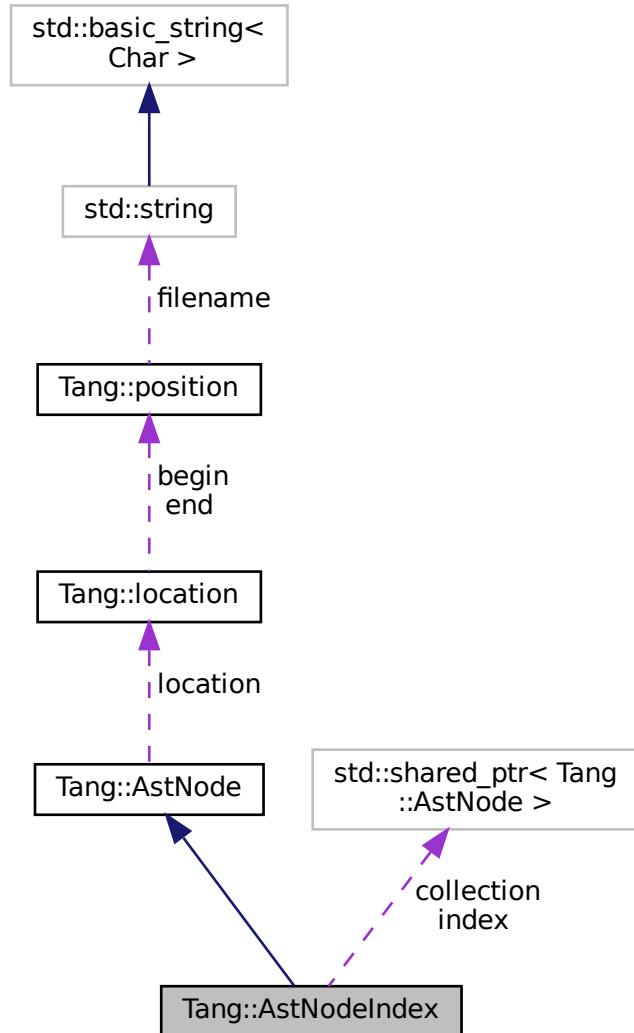
An [AstNode](#) that represents an index into a collection.

```
#include <astNodeIndex.hpp>
```

Inheritance diagram for Tang::AstNodeIndex:



Collaboration diagram for Tang::AstNodeIndex:



Public Types

- enum **PreprocessState** : int { **Default** = 0 , **IsAssignment** = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- AstNodeIndex** (**std::shared_ptr< AstNode > collection**, **std::shared_ptr< AstNode > index**, **Tang::location**)
The constructor.
- virtual std::string dump** (**std::string indent=""**) **const override**
Return a string that describes the contents of the node.

- virtual void `compile (Tang::Program &program)` const override
Compile the ast of the provided `Tang::Program`.
- virtual void `compilePreprocess (Program &program, PreprocessState state)` const override
Run any preprocess analysis needed before compilation.
- const std::shared_ptr< const `AstNode` > `getCollection ()` const
Return a shared pointer to the `AstNode` serving as the Collection.
- const std::shared_ptr< const `AstNode` > `getIndex ()` const
Return a shared pointer to the `AstNode` serving as the Index.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- std::shared_ptr< `AstNode` > `collection`
The collection into which we will index.
- std::shared_ptr< `AstNode` > `index`
The index expression.

5.17.1 Detailed Description

An `AstNode` that represents an index into a collection.

5.17.2 Member Enumeration Documentation

5.17.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.17.3 Constructor & Destructor Documentation

5.17.3.1 AstNodeIndex()

```
AstNodeIndex::AstNodeIndex (
    std::shared_ptr< AstNode > collection,
    std::shared_ptr< AstNode > index,
    Tang::location location )
```

The constructor.

Parameters

<i>collection</i>	The collection into which we will index.
<i>index</i>	The index expression.
<i>location</i>	The location associated with the expression.

5.17.4 Member Function Documentation

5.17.4.1 compile()

```
void AstNodeIndex::compile (
    Tang::Program & program ) const [override], [virtual]
```

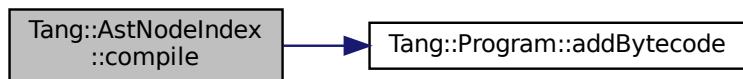
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.17.4.2 compilePreprocess()

```
void AstNodeIndex::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.17.4.3 dump()

```
string AstNodeIndex::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

5.17.4.4 getCollection()

```
const std::shared_ptr< const AstNode > AstNodeIndex::getCollection ( ) const
```

Return a shared pointer to the [AstNode](#) serving as the Collection.

Returns

The collection into which we will index.

5.17.4.5 getIndex()

```
const std::shared_ptr< const AstNode > AstNodeIndex::getIndex ( ) const
```

Return a shared pointer to the [AstNode](#) serving as the Index.

Returns

The index expression.

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

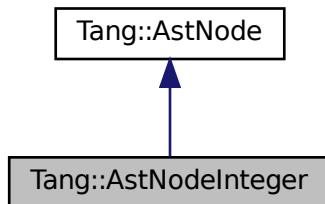
- [include/astNodeIndex.hpp](#)
- [src/astNodeIndex.cpp](#)

5.18 Tang::AstNodeInteger Class Reference

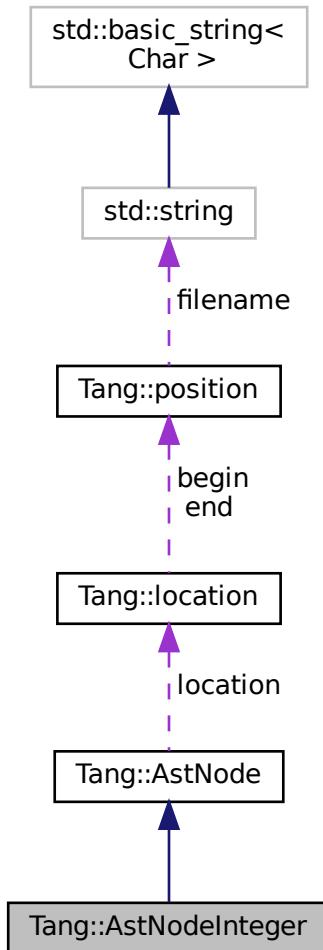
An [AstNode](#) that represents an integer literal.

```
#include <astNodeInteger.hpp>
```

Inheritance diagram for Tang::AstNodeInteger:



Collaboration diagram for Tang::AstNodeInteger:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeInteger (Tang::integer_t number, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`
Compile the ast of the provided `Tang::Program`.
- `virtual void compilePreprocess (Program &program, PreprocessState state) const`
Run any preprocess analysis needed before compilation.

Protected Attributes

- [Tang::location location](#)
The location associated with this node.

Private Attributes

- [Tang::integer_t val](#)
The integer value being stored.

5.18.1 Detailed Description

An [AstNode](#) that represents an integer literal.

Integers are represented by the `Tang::integer_t` type, and so are limited in range by that of the underlying type.

5.18.2 Member Enumeration Documentation

5.18.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	AstNode is part of an assignment expression.

5.18.3 Constructor & Destructor Documentation

5.18.3.1 AstNodeInteger()

```
AstNodeInteger::AstNodeInteger (
    Tang::integer_t number,
    Tang::location location )
```

The constructor.

Parameters

<i>number</i>	The number to represent.
<i>location</i>	The location associated with the expression.

5.18.4 Member Function Documentation

5.18.4.1 compile()

```
void AstNodeInteger::compile (
    Tang::Program & program ) const [override], [virtual]
```

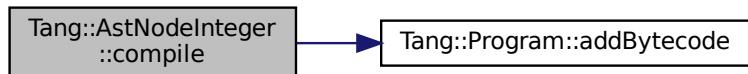
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.18.4.2 compilePreprocess()

```
void AstNode::compilePreprocess (
    Program & program,
    PreprocessState state ) const [virtual], [inherited]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUse](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#), [Tang::AstNodeMap](#), [Tang::AstNodeLibrary](#), [Tang::AstNodeIndex](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFunctionDeclaration](#), [Tang::AstNodeFunctionCall](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeCast](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), [Tang::AstNodeAssign](#), and [Tang::AstNodeArray](#).

5.18.4.3 `dump()`

```
string AstNodeInteger::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

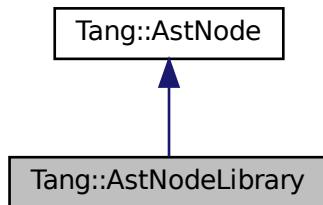
- [include/astNodeInteger.hpp](#)
- [src/astNodeInteger.cpp](#)

5.19 Tang::AstNodeLibrary Class Reference

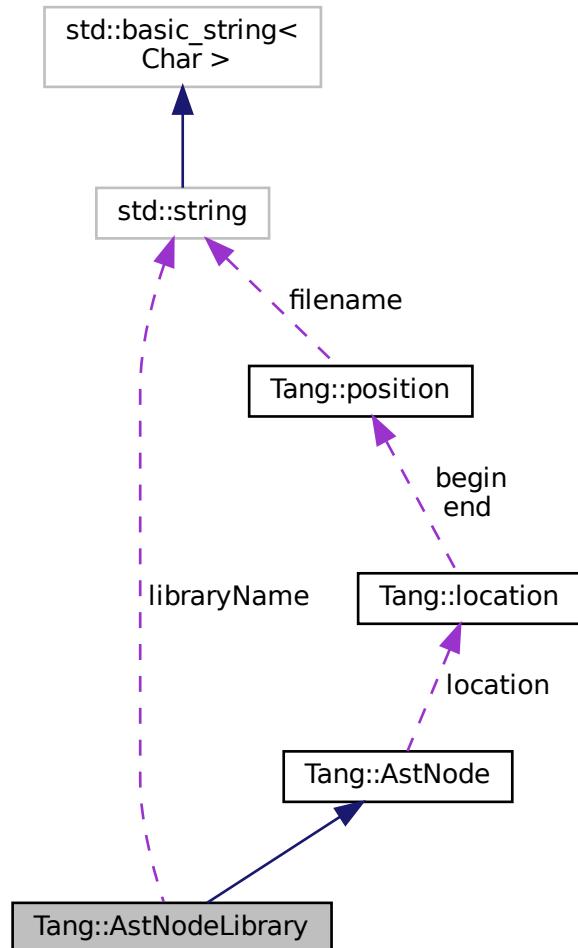
An [AstNode](#) that represents an identifier.

```
#include <astNodeLibrary.hpp>
```

Inheritance diagram for Tang::AstNodeLibrary:



Collaboration diagram for Tang::AstNodeLibrary:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeLibrary (const std::string &libraryName, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`
Compile the ast of the provided `Tang::Program`.
- `virtual void compilePreprocess (Program &program, PreprocessState state) const override`
Run any preprocess analysis needed before compilation.

Public Attributes

- std::string `libraryName`

The library name.

Protected Attributes

- Tang::location `location`

The location associated with this node.

5.19.1 Detailed Description

An `AstNode` that represents an identifier.

Library names are represented by a string.

5.19.2 Member Enumeration Documentation

5.19.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.19.3 Constructor & Destructor Documentation

5.19.3.1 AstNodeLibrary()

```
AstNodeLibrary::AstNodeLibrary (
    const std::string & libraryName,
    Tang::location location )
```

The constructor.

Parameters

<i>expression</i>	The library expression.
<i>location</i>	The location associated with the expression.

5.19.4 Member Function Documentation

5.19.4.1 compile()

```
void AstNodeLibrary::compile (
    Tang::Program & program ) const [override], [virtual]
```

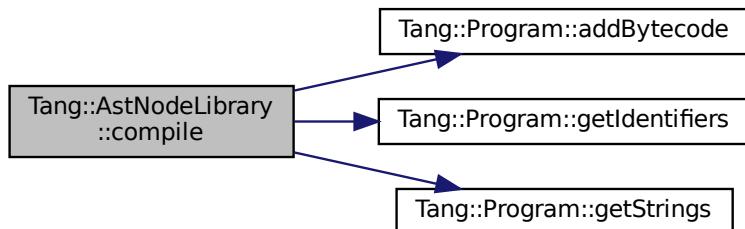
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.19.4.2 compilePreprocess()

```
void AstNodeLibrary::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

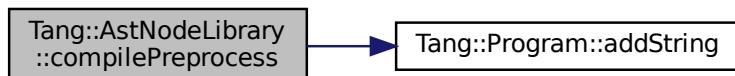
Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.19.4.3 dump()

```
string AstNodeLibrary::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

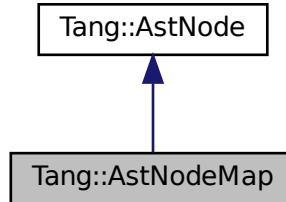
- [include/astNodeLibrary.hpp](#)
- [src/astNodeLibrary.cpp](#)

5.20 Tang::AstNodeMap Class Reference

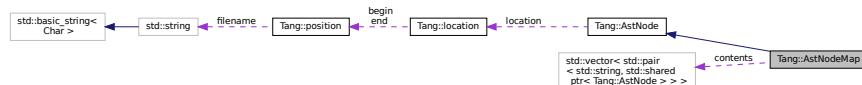
An [AstNode](#) that represents a map literal.

```
#include <astNodeMap.hpp>
```

Inheritance diagram for Tang::AstNodeMap:



Collaboration diagram for Tang::AstNodeMap:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeMap (std::vector< std::pair< std::string, std::shared_ptr< Tang::AstNode >>> >>> contents, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`
Compile the ast of the provided Tang::Program.
- `virtual void compilePreprocess (Program &program, PreprocessState state) const override`
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `std::vector< std::pair< std::string, std::shared_ptr< Tang::AstNode >>> >>> contents`
The contents of the array.

5.20.1 Detailed Description

An [AstNode](#) that represents a map literal.

Keys can only be strings.

5.20.2 Member Enumeration Documentation

5.20.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	AstNode is part of an assignment expression.

5.20.3 Constructor & Destructor Documentation

5.20.3.1 AstNodeMap()

```
AstNodeMap::AstNodeMap (
    std::vector< std::pair< std::string, std::shared_ptr< Tang::AstNode >>> contents,
    Tang::location location )
```

The constructor.

Parameters

<i>contents</i>	The contents of the map.
<i>location</i>	The location associated with the expression.

5.20.4 Member Function Documentation

5.20.4.1 compile()

```
void AstNodeMap::compile (
    Tang::Program & program ) const [override], [virtual]
```

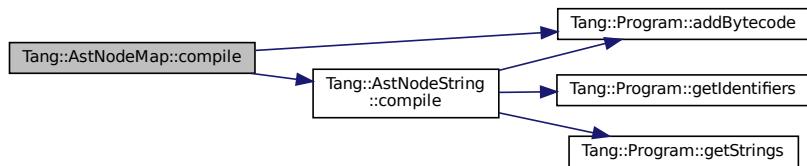
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.20.4.2 compilePreprocess()

```
void AstNodeMap::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.20.4.3 dump()

```
string AstNodeMap::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

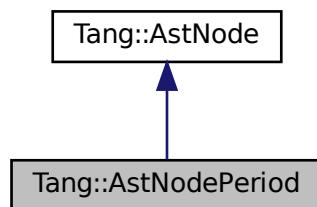
- [include/astNodeMap.hpp](#)
- [src/astNodeMap.cpp](#)

5.21 Tang::AstNodePeriod Class Reference

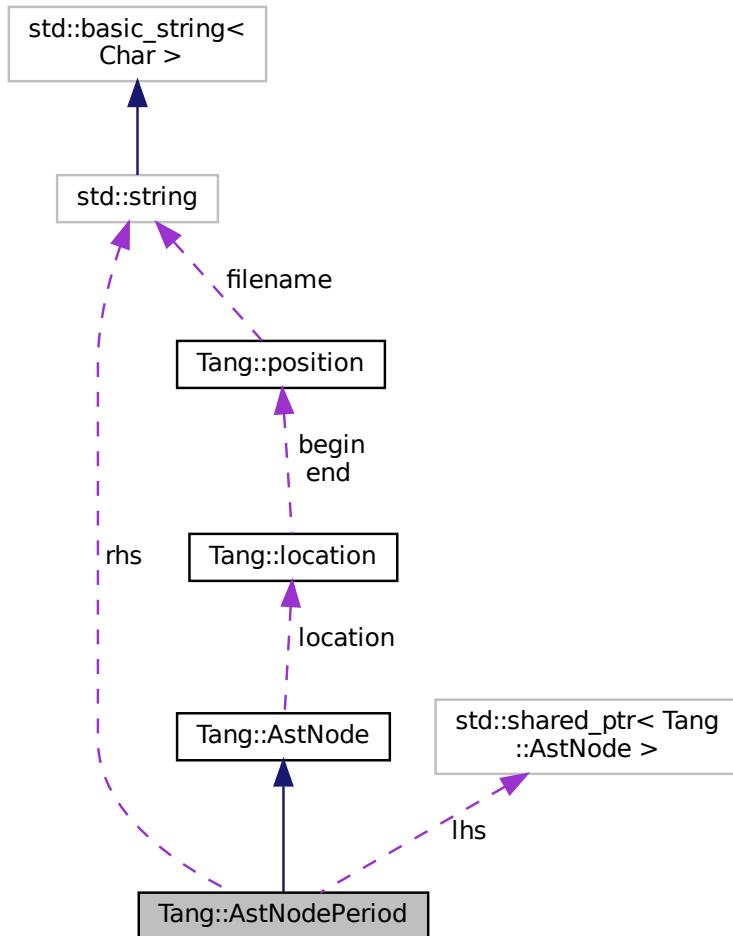
An [AstNode](#) that represents a member access (period) into an object.

```
#include <astNodePeriod.hpp>
```

Inheritance diagram for Tang::AstNodePeriod:



Collaboration diagram for Tang::AstNodePeriod:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodePeriod (std::shared_ptr< AstNode > lhs, std::string rhs, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`
Compile the ast of the provided `Tang::Program`.
- `virtual void compilePreprocess (Program &program, PreprocessState state) const override`
Run any preprocess analysis needed before compilation.

Protected Attributes

- [Tang::location location](#)
The location associated with this node.

Private Attributes

- [std::shared_ptr< AstNode > lhs](#)
The lhs into which we will rhs.
- [std::string rhs](#)
The rhs expression.

5.21.1 Detailed Description

An [AstNode](#) that represents a member access (period) into an object.

5.21.2 Member Enumeration Documentation

5.21.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	AstNode is part of an assignment expression.

5.21.3 Constructor & Destructor Documentation

5.21.3.1 AstNodePeriod()

```
AstNodePeriod::AstNodePeriod (
    std::shared_ptr< AstNode > lhs,
    std::string rhs,
    Tang::location location )
```

The constructor.

Parameters

<i>lhs</i>	The lhs on which the member access will be performed
<i>rhs</i>	The rhs identifier.
<i>location</i>	The location associated with the expression.

5.21.4 Member Function Documentation

5.21.4.1 `compile()`

```
void AstNodePeriod::compile (
    Tang::Program & program ) const [override], [virtual]
```

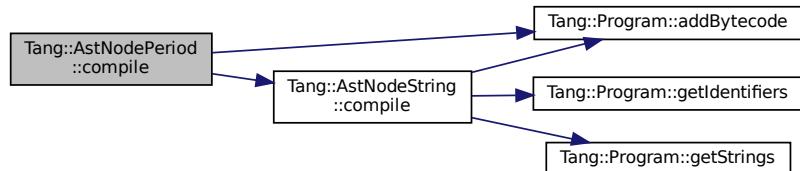
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:

5.21.4.2 `compilePreprocess()`

```
void AstNodePeriod::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.21.4.3 dump()

```
string AstNodePeriod::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

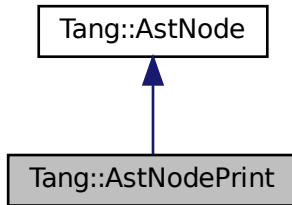
- [include/astNodePeriod.hpp](#)
- [src/astNodePeriod.cpp](#)

5.22 Tang::AstNodePrint Class Reference

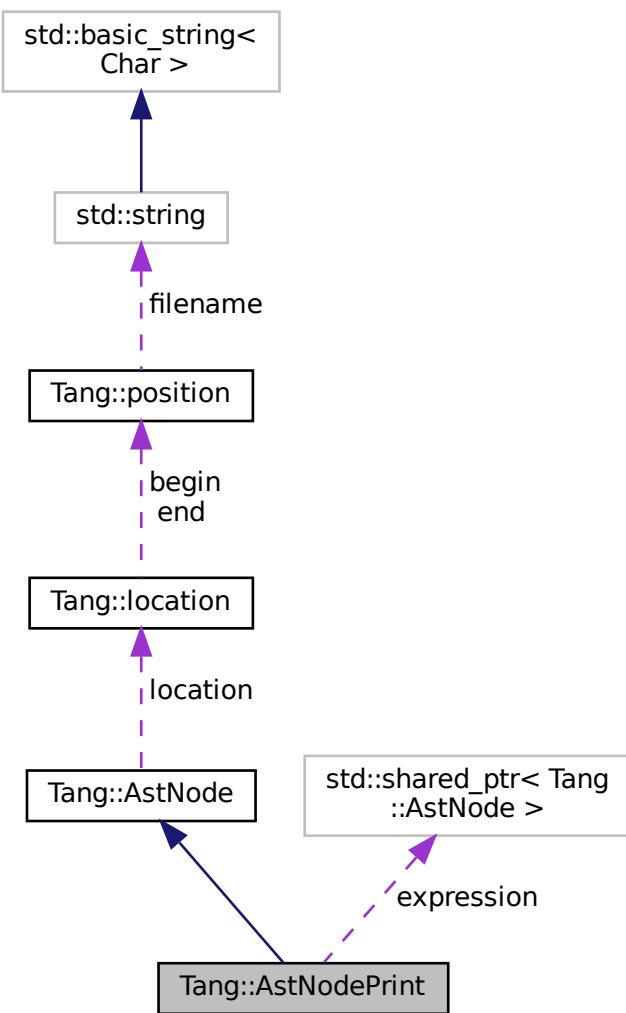
An [AstNode](#) that represents a print typeeration.

```
#include <astNodePrint.hpp>
```

Inheritance diagram for Tang::AstNodePrint:



Collaboration diagram for Tang::AstNodePrint:



Public Types

- enum `Type` { `Default` }
The type of `print()` requested.
- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Public Member Functions

- `AstNodePrint (Type type, std::shared_ptr< AstNode > expression, Tang::location location)`
The constructor.
- virtual std::string `dump` (std::string `indent`= "") const override
Return a string that describes the contents of the node.
- virtual void `compile` (Tang::Program &program) const override
Compile the ast of the provided Tang::Program.
- virtual void `compilePreprocess` (Program &program, PreprocessState state) const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `Type type`
The type of `print()` being requested.
- `std::shared_ptr< AstNode > expression`
The expression to be printed.

5.22.1 Detailed Description

An `AstNode` that represents a print typeeration.

5.22.2 Member Enumeration Documentation

5.22.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	AstNode is part of an assignment expression.

5.22.2.2 Type

```
enum Tang::AstNodePrint::Type
```

The type of print() requested.

Enumerator

Default	Use the default print.
---------	------------------------

5.22.3 Constructor & Destructor Documentation**5.22.3.1 [AstNodePrint\(\)](#)**

```
AstNodePrint::AstNodePrint (
    Type type,
    std::shared_ptr< AstNode > expression,
    Tang::location location )
```

The constructor.

Parameters

<i>type</i>	The Tang::AstNodePrint::Type being requested.
<i>expression</i>	The expression to be printed.
<i>location</i>	The location associated with the expression.

5.22.4 Member Function Documentation**5.22.4.1 [compile\(\)](#)**

```
void AstNodePrint::compile (
    Tang::Program & program ) const [override], [virtual]
```

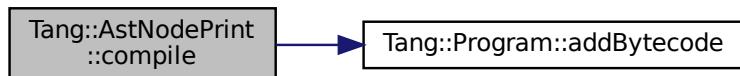
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.22.4.2 compilePreprocess()

```
void AstNodePrint::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.22.4.3 dump()

```
string AstNodePrint::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

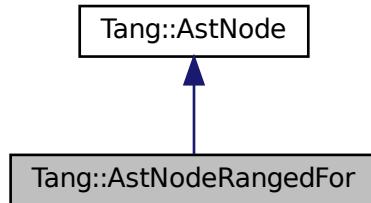
- [include/astNodePrint.hpp](#)
- [src/astNodePrint.cpp](#)

5.23 Tang::AstNodeRangedFor Class Reference

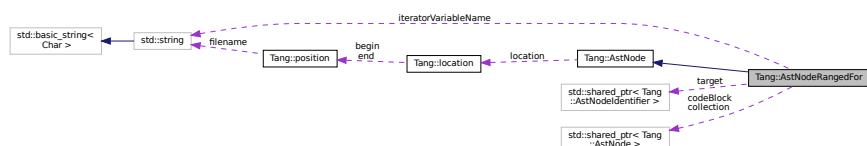
An [AstNode](#) that represents a ranged for() statement.

```
#include <astNodeRangedFor.hpp>
```

Inheritance diagram for Tang::AstNodeRangedFor:



Collaboration diagram for Tang::AstNodeRangedFor:



Public Types

- enum [PreprocessState](#) : int { [Default](#) = 0 , [IsAssignment](#) = 1 }

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Public Member Functions

- `AstNodeRangedFor (std::shared_ptr< AstNodelIdentifier > target, std::shared_ptr< AstNode > collection, std::shared_ptr< AstNode > codeBlock, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`
Compile the ast of the provided Tang::Program.
- `virtual void compilePreprocess (Program &program, PreprocessState state) const override`
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `std::shared_ptr< AstNodelIdentifier > target`
The target variable to hold the value for the current loop iteration.
- `std::shared_ptr< AstNode > collection`
The collection through which to iterate.
- `std::shared_ptr< AstNode > codeBlock`
The code block executed when the condition is true.
- `std::string iteratorVariableName`
The unique variable name that this iterator will use to persist its state on the stack.

5.23.1 Detailed Description

An `AstNode` that represents a ranged for() statement.

5.23.2 Member Enumeration Documentation

5.23.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.23.3 Constructor & Destructor Documentation

5.23.3.1 AstNodeRangedFor()

```
AstNodeRangedFor::AstNodeRangedFor (
    std::shared_ptr< AstNodeIdentifier > target,
    std::shared_ptr< AstNode > collection,
    std::shared_ptr< AstNode > codeBlock,
    Tang::location location )
```

The constructor.

Parameters

<i>target</i>	The target variable to hold the value for the current loop iteration.
<i>collection</i>	The collection through which to iterate.
<i>codeBlock</i>	The statement executed when the condition is true.
<i>location</i>	The location associated with the expression.

5.23.4 Member Function Documentation

5.23.4.1 compile()

```
void AstNodeRangedFor::compile (
    Tang::Program & program ) const [override], [virtual]
```

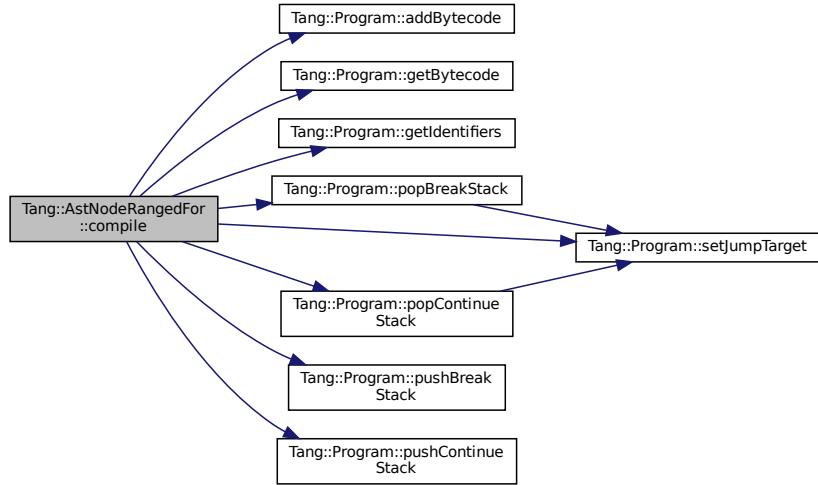
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.23.4.2 compilePreprocess()

```
void AstNodeRangedFor::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

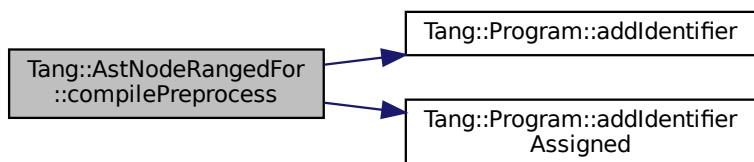
Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.23.4.3 `dump()`

```
string AstNodeRangedFor::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

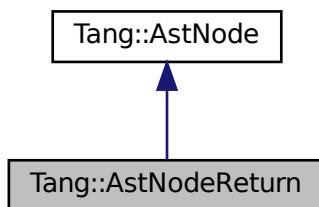
- [include/astNodeRangedFor.hpp](#)
- [src/astNodeRangedFor.cpp](#)

5.24 Tang::AstNodeReturn Class Reference

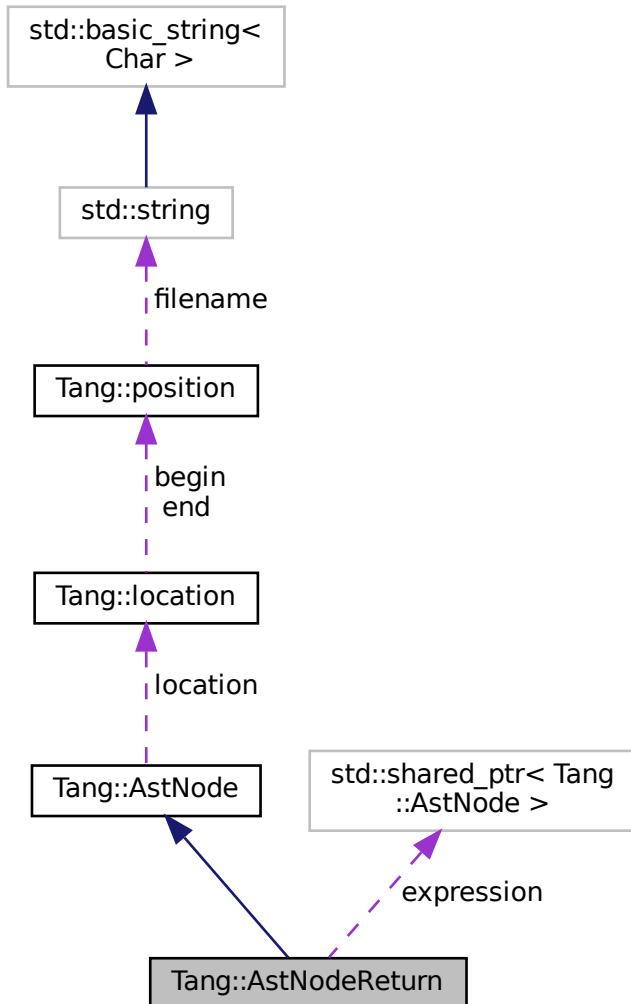
An [AstNode](#) that represents a `return` statement.

```
#include <astNodeReturn.hpp>
```

Inheritance diagram for Tang::AstNodeReturn:



Collaboration diagram for Tang::AstNodeReturn:



Public Types

- enum **PreprocessState** : int { **Default** = 0 , **IsAssignment** = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- AstNodeReturn** (`std::shared_ptr<AstNode> expression, Tang::location location)`
The constructor.
- virtual std::string **dump** (`std::string indent="")` const override
Return a string that describes the contents of the node.
- virtual void **compile** (`Tang::Program &program`) const override
Compile the ast of the provided `Tang::Program`.
- virtual void **compilePreprocess** (`Program &program, PreprocessState state`) const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `std::shared_ptr< AstNode > expression`
The expression to which the operation will be applied.

5.24.1 Detailed Description

An `AstNode` that represents a `return` statement.

5.24.2 Member Enumeration Documentation

5.24.2.1 PreprocessState

`enum Tang::AstNode::PreprocessState : int [inherited]`

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.24.3 Constructor & Destructor Documentation

5.24.3.1 AstNodeReturn()

```
AstNodeReturn::AstNodeReturn (
    std::shared_ptr< AstNode > expression,
    Tang::location location )
```

The constructor.

Parameters

<code>expression</code>	The expression to be returned.
<code>location</code>	The location associated with the return statement.

5.24.4 Member Function Documentation

5.24.4.1 compile()

```
void AstNodeReturn::compile (
    Tang::Program & program ) const [override], [virtual]
```

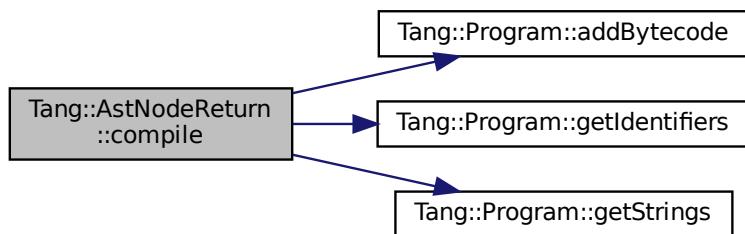
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.24.4.2 compilePreprocess()

```
void AstNodeReturn::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.24.4.3 `dump()`

```
string AstNodeReturn::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

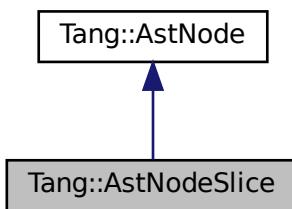
- [include/astNodeReturn.hpp](#)
- [src/astNodeReturn.cpp](#)

5.25 Tang::AstNodeSlice Class Reference

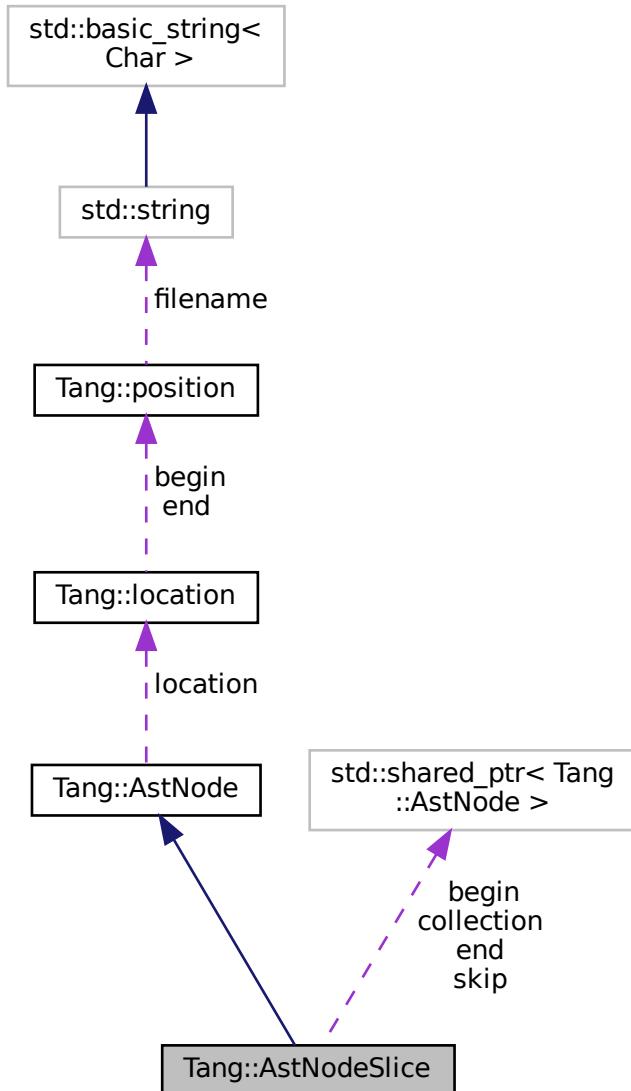
An [AstNode](#) that represents a ternary expression.

```
#include <astNodeSlice.hpp>
```

Inheritance diagram for Tang::AstNodeSlice:



Collaboration diagram for Tang::AstNodeSlice:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeSlice (std::shared_ptr< AstNode > collection, std::shared_ptr< AstNode > begin, std::shared_ptr< AstNode > end, std::shared_ptr< AstNode > slice, Tang::location location)`
- The constructor.*

- virtual std::string `dump` (std::string `indent=""`) const override
Return a string that describes the contents of the node.
- virtual void `compile` (`Tang::Program` &`program`) const override
Compile the ast of the provided `Tang::Program`.
- virtual void `compilePreprocess` (`Program` &`program`, `PreprocessState` `state`) const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location` `location`
The location associated with this node.

Private Attributes

- `std::shared_ptr<AstNode>` `collection`
The collection which will be sliced.
- `std::shared_ptr<AstNode>` `begin`
The begin index position of the slice.
- `std::shared_ptr<AstNode>` `end`
The end index position of the slice.
- `std::shared_ptr<AstNode>` `skip`
The skip index position of the slice.

5.25.1 Detailed Description

An `AstNode` that represents a ternary expression.

5.25.2 Member Enumeration Documentation

5.25.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

<code>Default</code>	The default state.
<code>IsAssignment</code>	<code>AstNode</code> is part of an assignment expression.

5.25.3 Constructor & Destructor Documentation

5.25.3.1 AstNodeSlice()

```
AstNodeSlice::AstNodeSlice (
    std::shared_ptr< AstNode > collection,
    std::shared_ptr< AstNode > begin,
    std::shared_ptr< AstNode > end,
    std::shared_ptr< AstNode > slice,
    Tang::location location )
```

The constructor.

Parameters

<i>collection</i>	The collection which will be sliced.
<i>begin</i>	The begin index position of the slice.
<i>end</i>	The end index position of the slice.
<i>skip</i>	The skip index position of the slice.
<i>location</i>	The location associated with the expression.

5.25.4 Member Function Documentation

5.25.4.1 compile()

```
void AstNodeSlice::compile (
    Tang::Program & program ) const [override], [virtual]
```

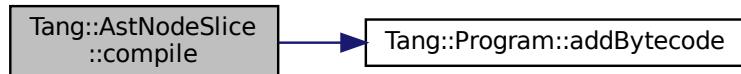
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.25.4.2 compilePreprocess()

```
void AstNodeSlice::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.25.4.3 dump()

```
string AstNodeSlice::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

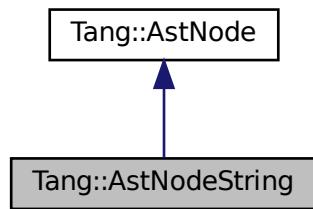
- [include/astNodeSlice.hpp](#)
- [src/astNodeSlice.cpp](#)

5.26 Tang::AstNodeString Class Reference

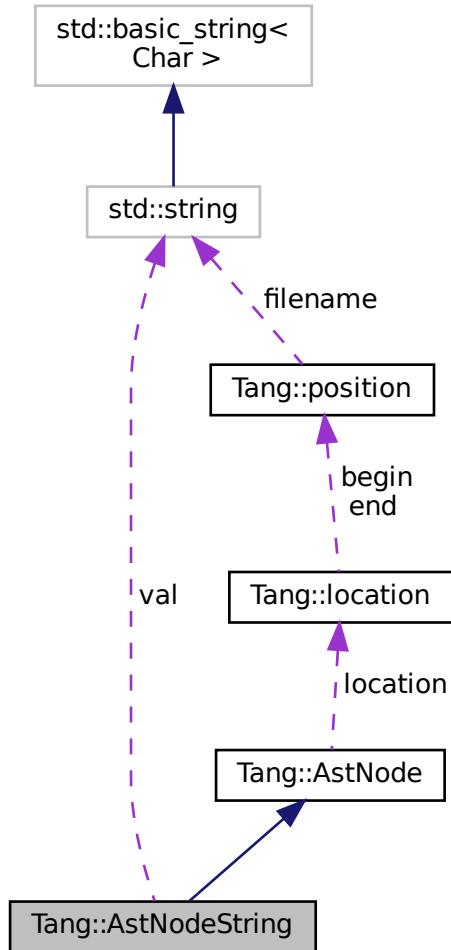
An [AstNode](#) that represents a string literal.

```
#include <astNodeString.hpp>
```

Inheritance diagram for Tang::AstNodeString:



Collaboration diagram for Tang::AstNodeString:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeString (const std::string &text, Tang::location location)`
Construct a Trusted string.
- `AstNodeString (const std::string &text, bool isTrusted, Tang::location location)`
Construct a string that is either Trusted or Untrusted.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.
- `virtual void compile (Tang::Program &program) const override`

Compile the ast of the provided Tang::Program.

- virtual void `compilePreprocess` (`Tang::Program` &`program`, `PreprocessState` `state`) const override
Run any preprocess analysis needed before compilation.
- void `compileLiteral` (`Tang::Program` &`program`) const
Compile the string and push it onto the stack.

Protected Attributes

- `Tang::location` `location`
The location associated with this node.

Private Attributes

- `std::string` `val`
The string value being stored.
- `bool` `isTrusted`
Whether or not the string is trusted.

5.26.1 Detailed Description

An `AstNode` that represents a string literal.

5.26.2 Member Enumeration Documentation

5.26.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.26.3 Constructor & Destructor Documentation

5.26.3.1 `AstNodeString()` [1/2]

```
AstNodeString::AstNodeString (
    const std::string & text,
    Tang::location location )
```

Construct a Trusted string.

Parameters

<i>text</i>	The string to represent.
<i>location</i>	The location associated with the expression.

5.26.3.2 `AstNodeString()` [2/2]

```
AstNodeString::AstNodeString (
    const std::string & text,
    bool isTrusted,
    Tang::location location )
```

Construct a string that is either Trusted or Untrusted.

Parameters

<i>text</i>	The string to represent.
<i>isTrusted</i>	Whether or not the string literal is trusted.
<i>location</i>	The location associated with the expression.

5.26.4 Member Function Documentation

5.26.4.1 `compile()`

```
void AstNodeString::compile (
    Tang::Program & program ) const [override], [virtual]
```

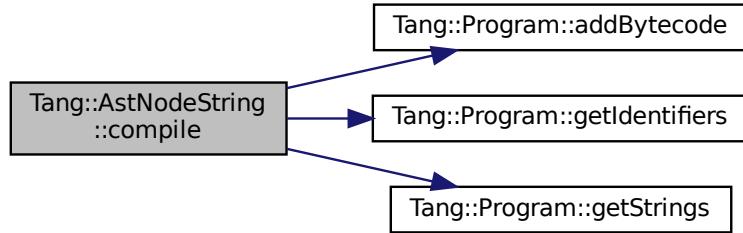
Compile the ast of the provided `Tang::Program`.

Parameters

<i>program</i>	The <code>Program</code> which will hold the generated Bytecode.
----------------	--

Reimplemented from `Tang::AstNode`.

Here is the call graph for this function:



5.26.4.2 compileLiteral()

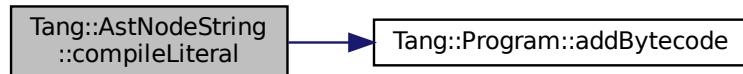
```
void AstNodeString::compileLiteral (
    Tang::Program & program ) const
```

Compile the string and push it onto the stack.

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Here is the call graph for this function:



5.26.4.3 compilePreprocess()

```
void AstNodeString::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

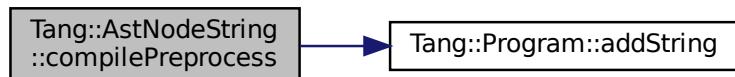
Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.26.4.4 `dump()`

```
string AstNodeString::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

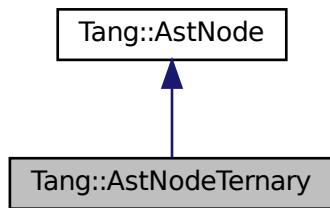
- [include/astNodeString.hpp](#)
- [src/astNodeString.cpp](#)

5.27 Tang::AstNodeTernary Class Reference

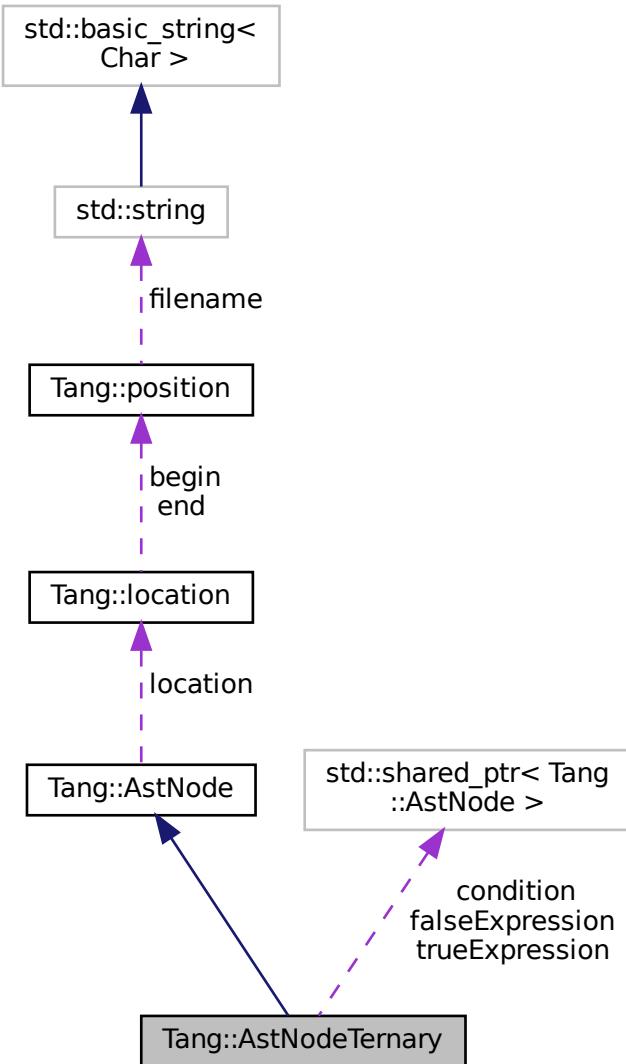
An [AstNode](#) that represents a ternary expression.

```
#include <astNodeTernary.hpp>
```

Inheritance diagram for Tang::AstNodeTernary:



Collaboration diagram for Tang::AstNodeTernary:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
- Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.*

Public Member Functions

- `AstNodeTernary (std::shared_ptr< AstNode > condition, std::shared_ptr< AstNode > trueExpression, std::shared_ptr< AstNode > falseExpression, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`

Return a string that describes the contents of the node.

- virtual void [compile](#) ([Tang::Program](#) &program) const override
Compile the ast of the provided [Tang::Program](#).
- virtual void [compilePreprocess](#) ([Program](#) &program, [PreprocessState](#) state) const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- [Tang::location](#) location
The location associated with this node.

Private Attributes

- std::shared_ptr< [AstNode](#) > condition
The expression which determines whether the trueExpression or falseExpression is executed.
- std::shared_ptr< [AstNode](#) > trueExpression
The expression executed when the condition is true.
- std::shared_ptr< [AstNode](#) > falseExpression
The expression executed when the condition is false.

5.27.1 Detailed Description

An [AstNode](#) that represents a ternary expression.

5.27.2 Member Enumeration Documentation

5.27.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	AstNode is part of an assignment expression.

5.27.3 Constructor & Destructor Documentation

5.27.3.1 AstNodeTernary()

```
AstNodeTernary::AstNodeTernary (
    std::shared_ptr< AstNode > condition,
    std::shared_ptr< AstNode > trueExpression,
    std::shared_ptr< AstNode > falseExpression,
    Tang::location location )
```

The constructor.

Parameters

<i>condition</i>	The expression which determines whether the trueExpression or falseExpression is executed.
<i>trueExpression</i>	The expression executed when the condition is true.
<i>falseExpression</i>	The expression executed when the condition is false.
<i>location</i>	The location associated with the expression.

5.27.4 Member Function Documentation

5.27.4.1 compile()

```
void AstNodeTernary::compile (
    Tang::Program & program ) const [override], [virtual]
```

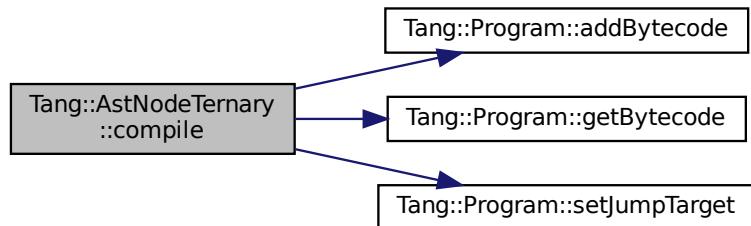
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.27.4.2 compilePreprocess()

```
void AstNodeTernary::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.27.4.3 dump()

```
string AstNodeTernary::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

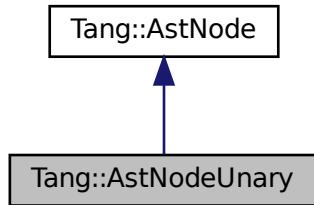
- [include/astNodeTernary.hpp](#)
- [src/astNodeTernary.cpp](#)

5.28 Tang::AstNodeUnary Class Reference

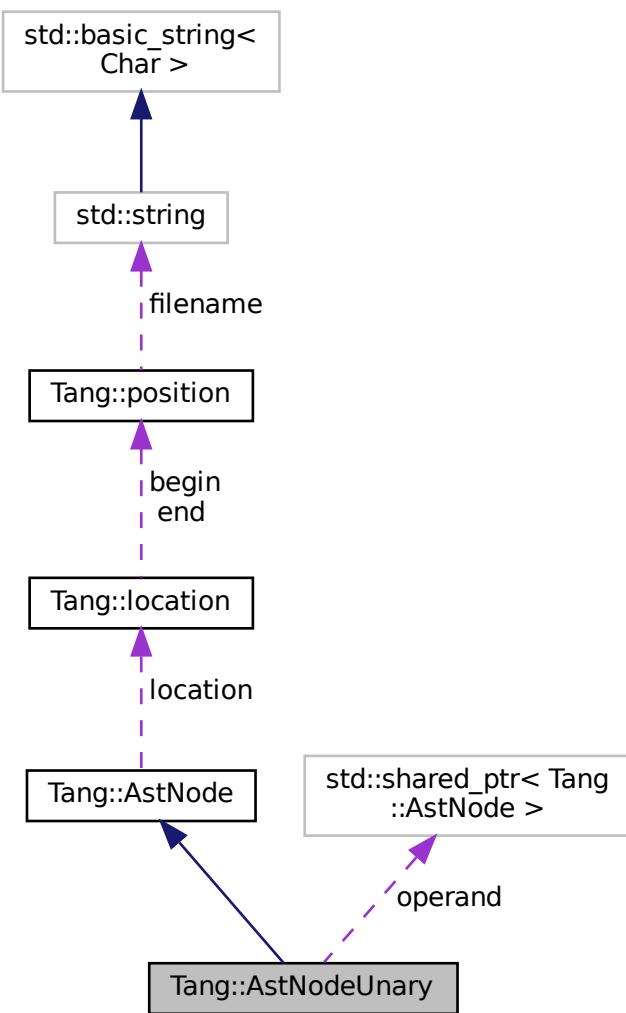
An [AstNode](#) that represents a unary negation.

```
#include <astNodeUnary.hpp>
```

Inheritance diagram for Tang::AstNodeUnary:



Collaboration diagram for Tang::AstNodeUnary:



Public Types

- enum `Operator` { `Negative` , `Not` }
The type of operation.
- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }
Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Public Member Functions

- `AstNodeUnary (Operator op, std::shared_ptr< AstNode > operand, Tang::location location)`
The constructor.
- virtual std::string `dump` (std::string indent="") const override
Return a string that describes the contents of the node.
- virtual void `compile` (Tang::Program &program) const override
Compile the ast of the provided Tang::Program.
- virtual void `compilePreprocess` (Program &program, PreprocessState state) const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `Operator op`
The operation which will be applied to the operand.
- `std::shared_ptr< AstNode > operand`
The operand to which the operation will be applied.

5.28.1 Detailed Description

An `AstNode` that represents a unary negation.

5.28.2 Member Enumeration Documentation

5.28.2.1 Operator

```
enum Tang::AstNodeUnary::Operator
```

The type of operation.

Enumerator

Negative	Compute the negative (-).
Not	Compute the logical not (!).

5.28.2.2 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	AstNode is part of an assignment expression.

5.28.3 Constructor & Destructor Documentation**5.28.3.1 AstNodeUnary()**

```
AstNodeUnary::AstNodeUnary (
    Operator op,
    std::shared_ptr< AstNode > operand,
    Tang::location location )
```

The constructor.

Parameters

<i>op</i>	The Tang::AstNodeUnary::Operator to apply to the operand.
<i>operand</i>	The expression to be operated on.
<i>location</i>	The location associated with the expression.

5.28.4 Member Function Documentation**5.28.4.1 compile()**

```
void AstNodeUnary::compile (
    Tang::Program & program ) const [override], [virtual]
```

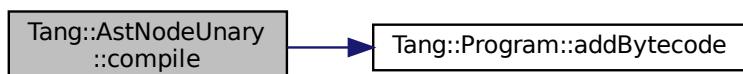
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.28.4.2 compilePreprocess()

```
void AstNodeUnary::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.28.4.3 dump()

```
string AstNodeUnary::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

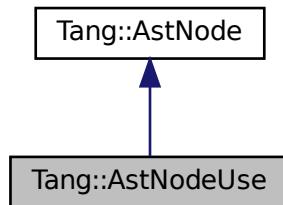
- [include/astNodeUnary.hpp](#)
- [src/astNodeUnary.cpp](#)

5.29 Tang::AstNodeUse Class Reference

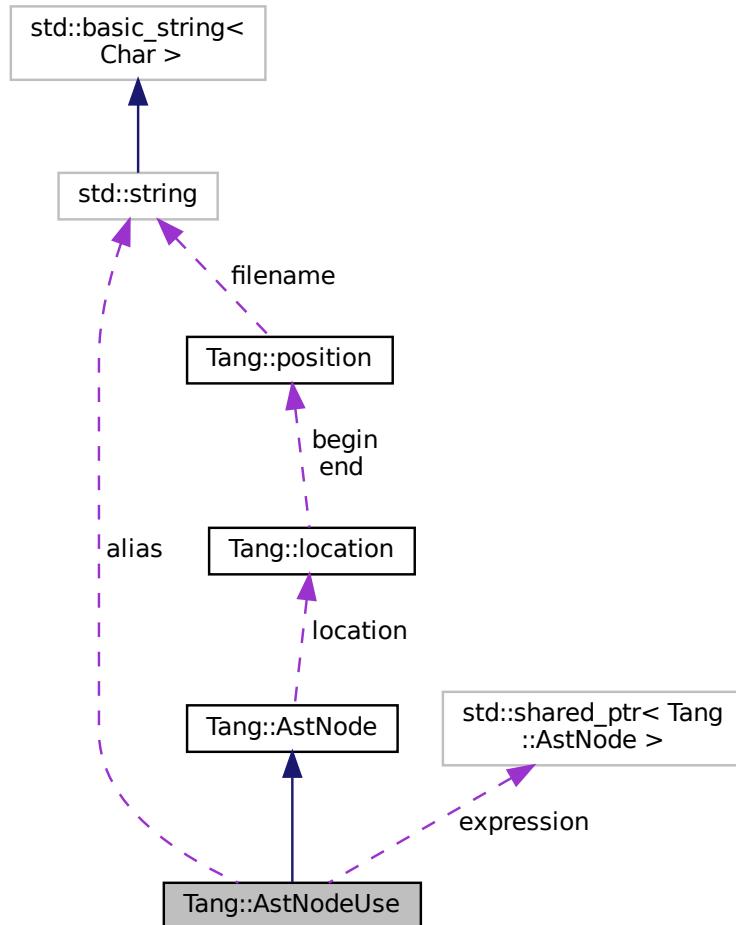
An [AstNode](#) that represents the inclusion of a library into the script.

```
#include <astNodeUse.hpp>
```

Inheritance diagram for Tang::AstNodeUse:



Collaboration diagram for Tang::AstNodeUse:



Public Types

- enum **PreprocessState** : int { **Default** = 0 , **IsAssignment** = 1 }
Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Public Member Functions

- AstNodeUse** (**std::shared_ptr<AstNode> expression**, **const std::string &alias**, **Tang::location location**)
The constructor.
- virtual std::string dump** (**std::string indent=""**) **const override**
Return a string that describes the contents of the node.
- virtual void compile** (**Tang::Program &program**) **const override**
Compile the ast of the provided [Tang::Program](#).
- virtual void compilePreprocess** (**Program &program**, **PreprocessState state**) **const override**
Run any preprocess analysis needed before compilation.

Public Attributes

- std::string **alias**
The alias to use for the library expression.
- std::shared_ptr< **AstNode** > **expression**
The library expression.

Protected Attributes

- **Tang::location location**
The location associated with this node.

5.29.1 Detailed Description

An **AstNode** that represents the inclusion of a library into the script.

A library or the library attributes will be represented by the **alias** within the script.

5.29.2 Member Enumeration Documentation

5.29.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	AstNode is part of an assignment expression.

5.29.3 Constructor & Destructor Documentation

5.29.3.1 AstNodeUse()

```
AstNodeUse::AstNodeUse (
    std::shared_ptr< AstNode > expression,
    const std::string & alias,
    Tang::location location )
```

The constructor.

Parameters

<i>expression</i>	The library expression.
<i>alias</i>	An alias used to access the library expression within the script.
<i>location</i>	The location associated with the expression.

5.29.4 Member Function Documentation

5.29.4.1 **compile()**

```
void AstNodeUse::compile (
    Tang::Program & program ) const [override], [virtual]
```

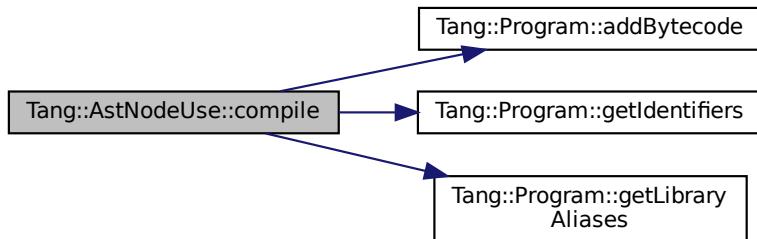
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:

5.29.4.2 **compilePreprocess()**

```
void AstNodeUse::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

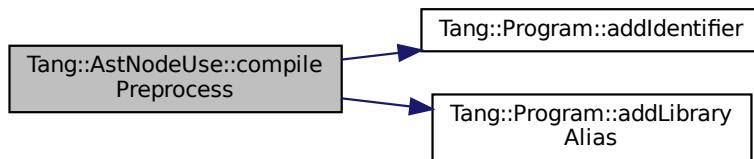
Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.29.4.3 dump()

```
string AstNodeUse::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

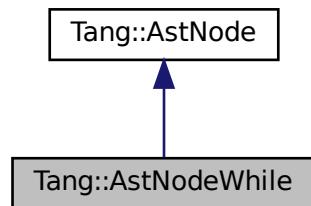
- [include/astNodeUse.hpp](#)
- [src/astNodeUse.cpp](#)

5.30 Tang::AstNodeWhile Class Reference

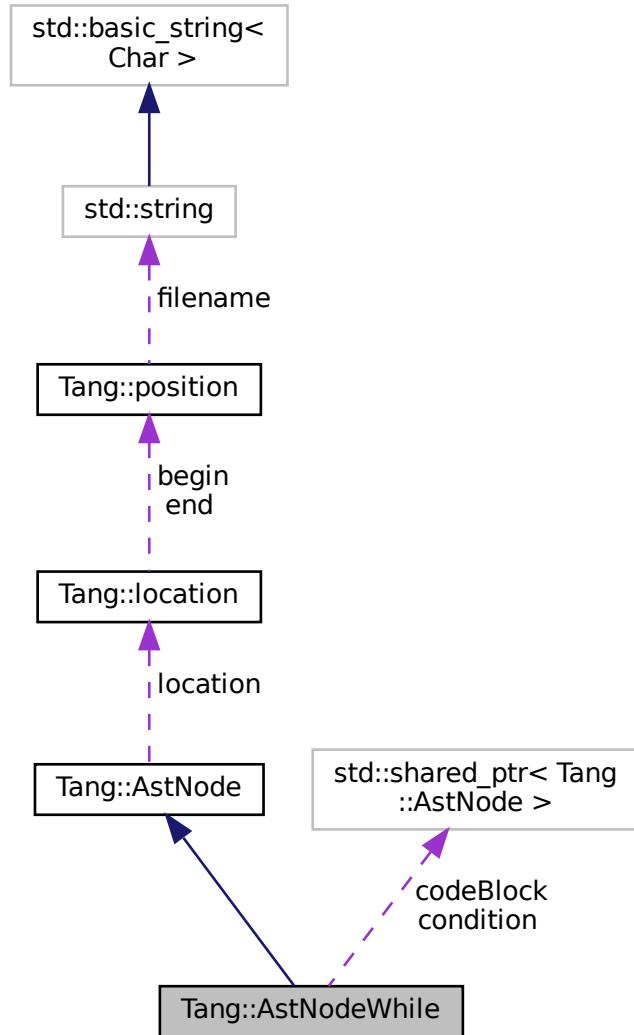
An [AstNode](#) that represents a while statement.

```
#include <astNodeWhile.hpp>
```

Inheritance diagram for Tang::AstNodeWhile:



Collaboration diagram for Tang::AstNodeWhile:



Public Types

- enum `PreprocessState` : int { `Default` = 0 , `IsAssignment` = 1 }

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Public Member Functions

- `AstNodeWhile (std::shared_ptr< AstNode > condition, std::shared_ptr< AstNode > codeBlock, Tang::location location)`
The constructor.
- `virtual std::string dump (std::string indent="") const override`
Return a string that describes the contents of the node.

- virtual void `compile (Tang::Program &program)` const override
Compile the ast of the provided `Tang::Program`.
- virtual void `compilePreprocess (Program &program, PreprocessState state)` const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- `Tang::location location`
The location associated with this node.

Private Attributes

- `std::shared_ptr< AstNode > condition`
The expression which determines whether or not the code block will continue to be executed.
- `std::shared_ptr< AstNode > codeBlock`
The code block executed when the condition is true.

5.30.1 Detailed Description

An `AstNode` that represents a while statement.

5.30.2 Member Enumeration Documentation

5.30.2.1 PreprocessState

```
enum Tang::AstNode::PreprocessState : int [inherited]
```

Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Enumerator

Default	The default state.
IsAssignment	<code>AstNode</code> is part of an assignment expression.

5.30.3 Constructor & Destructor Documentation

5.30.3.1 AstNodeWhile()

```
AstNodeWhile::AstNodeWhile (
    std::shared_ptr< AstNode > condition,
```

```
std::shared_ptr< AstNode > codeBlock,
Tang::location location )
```

The constructor.

Parameters

<i>condition</i>	The expression which determines whether the thenBlock or elseBlock is executed.
<i>codeBlock</i>	The statement executed when the condition is true.
<i>location</i>	The location associated with the expression.

5.30.4 Member Function Documentation

5.30.4.1 compile()

```
void AstNodeWhile::compile (
    Tang::Program & program ) const [override], [virtual]
```

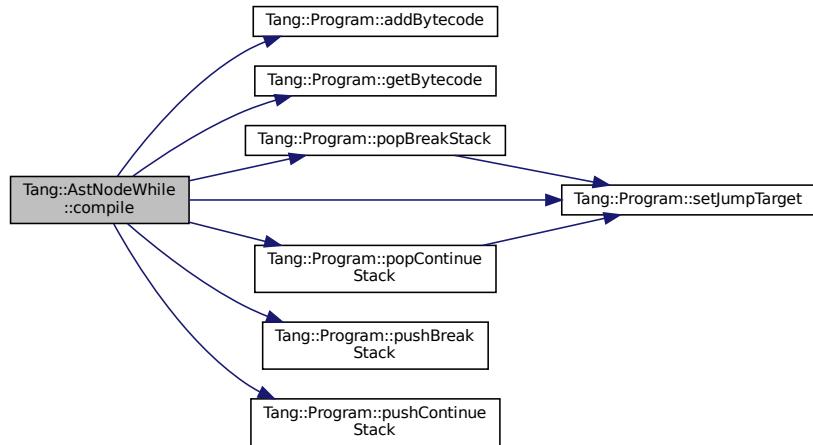
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Program which will hold the generated Bytecode.
----------------	---

Reimplemented from [Tang::AstNode](#).

Here is the call graph for this function:



5.30.4.2 compilePreprocess()

```
void AstNodeWhile::compilePreprocess (
    Program & program,
    PreprocessState state ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
<i>state</i>	Any preprocess flags that need to be considered.

Reimplemented from [Tang::AstNode](#).

5.30.4.3 dump()

```
string AstNodeWhile::dump (
    std::string indent = "" ) const [override], [virtual]
```

Return a string that describes the contents of the node.

Parameters

<i>indent</i>	A string used to indent the dump.
---------------	-----------------------------------

Returns

The value as a string.

Reimplemented from [Tang::AstNode](#).

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

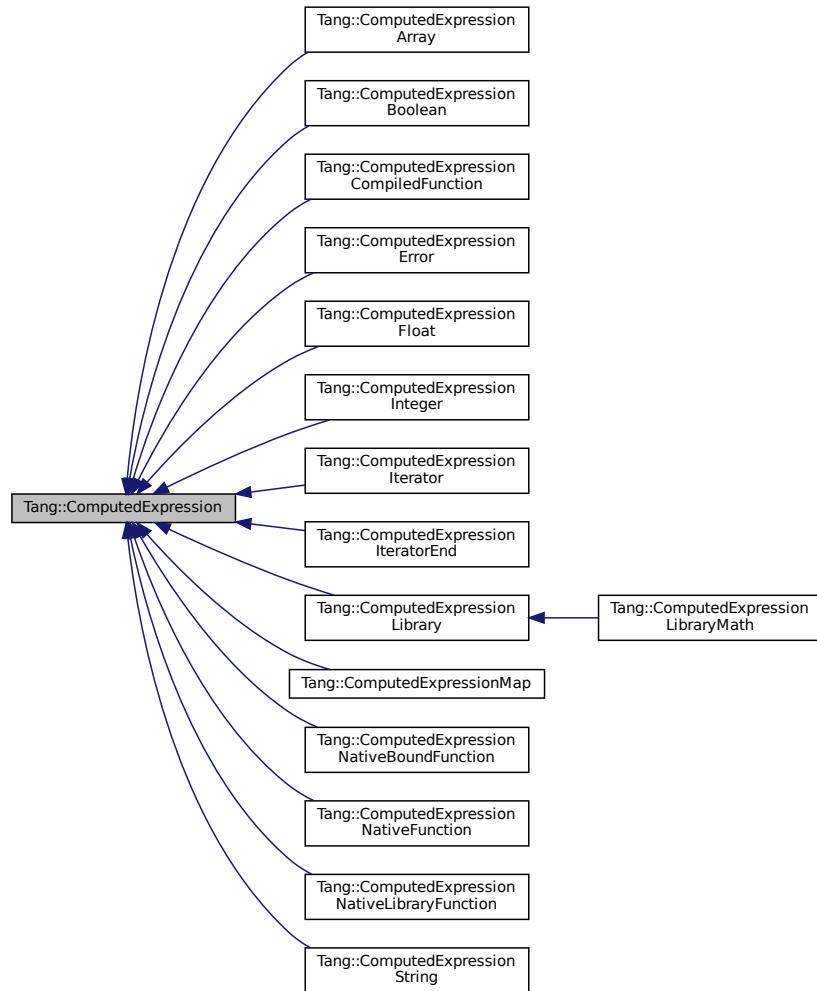
- [include/astNodeWhile.hpp](#)
- [src/astNodeWhile.cpp](#)

5.31 Tang::ComputedExpression Class Reference

Represents the result of a computation that has been executed.

```
#include <computedExpression.hpp>
```

Inheritance diagram for Tang::ComputedExpression:



Public Member Functions

- `virtual ~ComputedExpression ()`
The object destructor.
- `virtual std::string dump () const`
Output the contents of the `ComputedExpression` as a string.
- `virtual std::string __asCode () const`
Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- `virtual bool isCopyNeeded () const`
Determine whether or not a copy is needed.
- `virtual GarbageCollected makeCopy () const`
Make a copy of the `ComputedExpression` (recursively, if appropriate).
- `virtual bool is_equal (const Tang::integer_t &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const Tang::float_t &val) const`
Check whether or not the computed expression is equal to another value.

- virtual bool `is_equal` (const bool &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const std::string &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const Error &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual `GarbageCollected __assign_index` (const `GarbageCollected` &index, const `GarbageCollected` &value)
Perform an index assignment to the supplied value.
- virtual `GarbageCollected __add` (const `GarbageCollected` &rhs) const
Compute the result of adding this value and the supplied value.
- virtual `GarbageCollected __subtract` (const `GarbageCollected` &rhs) const
Compute the result of subtracting this value and the supplied value.
- virtual `GarbageCollected __multiply` (const `GarbageCollected` &rhs) const
Compute the result of multiplying this value and the supplied value.
- virtual `GarbageCollected __divide` (const `GarbageCollected` &rhs) const
Compute the result of dividing this value and the supplied value.
- virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __negative` () const
Compute the result of negating this value.
- virtual `GarbageCollected __not` () const
Compute the logical not of this value.
- virtual `GarbageCollected __lessThan` (const `GarbageCollected` &rhs) const
Compute the "less than" comparison.
- virtual `GarbageCollected __equal` (const `GarbageCollected` &rhs) const
Perform an equality test.
- virtual `GarbageCollected __period` (const `GarbageCollected` &member, std::shared_ptr<`TangBase`> &tang) const
Perform a member access (period) operation.
- virtual `GarbageCollected __index` (const `GarbageCollected` &index) const
Perform an index operation.
- virtual `GarbageCollected __slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const
Perform a slice operation.
- virtual `GarbageCollected __getIterator` (const `GarbageCollected` &collection) const
Get an iterator for the expression.
- virtual `GarbageCollected __iteratorNext` (size_t index=0) const
Get the next iterative value.
- virtual `GarbageCollected __integer` () const
Perform a type cast to integer.
- virtual `GarbageCollected __float` () const
Perform a type cast to float.
- virtual `GarbageCollected __boolean` () const
Perform a type cast to boolean.
- virtual `GarbageCollected __string` () const
Perform a type cast to string.

5.31.1 Detailed Description

Represents the result of a computation that has been executed.

By default, it will represent a NULL value.

5.31.2 Member Function Documentation

5.31.2.1 __add()

```
GarbageCollected ComputedExpression::__add (
    const GarbageCollected & rhs ) const [virtual]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.31.2.2 __asCode()

```
string ComputedExpression::__asCode ( ) const [virtual]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.31.2.3 __assign_index()

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.31.2.4 `__boolean()`

```
GarbageCollected ComputedExpression::__boolean ( ) const [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.31.2.5 `__divide()`

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.31.2.6 `__equal()`

```
GarbageCollected ComputedExpression::__equal (
    const GarbageCollected & rhs ) const [virtual]
```

Perform an equality test.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to compare against.
------------	---

Returns

The result of the the operation.

Reimplemented in `Tang::ComputedExpressionString`, `Tang::ComputedExpressionNativeLibraryFunction`, `Tang::ComputedExpressionTang::ComputedExpressionNativeBoundFunction`, `Tang::ComputedExpressionInteger`, `Tang::ComputedExpressionFloat`, `Tang::ComputedExpressionError`, `Tang::ComputedExpressionCompiledFunction`, and `Tang::ComputedExpressionBoolean`.

5.31.2.7 `__float()`

```
GarbageCollected ComputedExpression::__float () const [virtual]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in `Tang::ComputedExpressionInteger`, `Tang::ComputedExpressionFloat`, `Tang::ComputedExpressionError`, and `Tang::ComputedExpressionBoolean`.

5.31.2.8 `__getIterator()`

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The <code>GarbageCollected</code> value that will serve as the collection through which to iterate.
-------------------	---

Reimplemented in `Tang::ComputedExpressionString`, `Tang::ComputedExpressionMap`, and `Tang::ComputedExpressionArray`.

5.31.2.9 __index()

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual]
```

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.31.2.10 __integer()

```
GarbageCollected ComputedExpression::__integer () const [virtual]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.31.2.11 __iteratorNext()

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.31.2.12 __lessThan()

```
GarbageCollected ComputedExpression::__lessThan (
    const GarbageCollected & rhs ) const [virtual]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.31.2.13 __modulo()

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.31.2.14 __multiply()

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.31.2.15 `__negative()`

```
GarbageCollected ComputedExpression::__negative ( ) const [virtual]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.31.2.16 `__not()`

```
GarbageCollected ComputedExpression::__not ( ) const [virtual]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.31.2.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual]
```

Perform a member access (period) operation.

Parameters

<i>member</i>	The member expression provided by the script.
---------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.31.2.18 __slice()

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.31.2.19 __string()

```
GarbageCollected ComputedExpression::__string ( ) const [virtual]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.31.2.20 __subtract()

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.31.2.21 dump()

```
string ComputedExpression::dump ( ) const [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionNativeLibraryFunction](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionLibrary](#), [Tang::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionIterator](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), [Tang::ComputedExpressionBoolean](#), and [Tang::ComputedExpressionArray](#).

5.31.2.22 is_equal() [1/6]

```
virtual bool Tang::ComputedExpression::is_equal ( const bool & val ) const [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.31.2.23 `is_equal()` [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.31.2.24 `is_equal()` [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

5.31.2.25 `is_equal()` [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.31.2.26 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.31.2.27 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.31.2.28 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual]
```

Determine whether or not a copy is needed.

Copying is only required for **ComputedExpressions** which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.31.2.29 **makeCopy()**

`GarbageCollected ComputedExpression::makeCopy () const [virtual]`

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionNativeLibraryFunction](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionLibraryMath](#), [Tang::ComputedExpressionLibrary](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), [Tang::ComputedExpressionBoolean](#), and [Tang::ComputedExpressionArray](#).

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

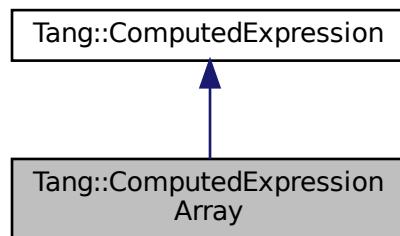
- `include/computedExpression.hpp`
- `src/computedExpression.cpp`

5.32 **Tang::ComputedExpressionArray** Class Reference

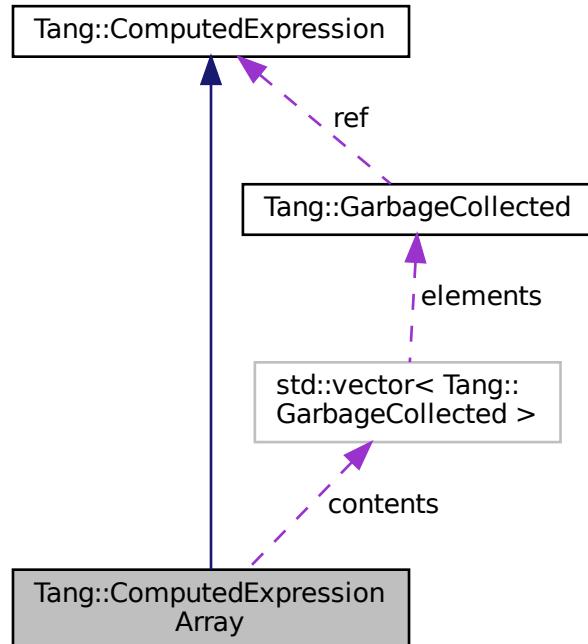
Represents an Array that is the result of a computation.

```
#include <computedExpressionArray.hpp>
```

Inheritance diagram for [Tang::ComputedExpressionArray](#):



Collaboration diagram for Tang::ComputedExpressionArray:



Public Member Functions

- **ComputedExpressionArray (std::vector< Tang::GarbageCollected > contents)**
Construct an Array result.
- virtual std::string **dump () const override**
Output the contents of the [ComputedExpression](#) as a string.
- virtual bool **isCopyNeeded () const override**
Determine whether or not a copy is needed.
- **GarbageCollected makeCopy () const override**
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual **GarbageCollected __index (const GarbageCollected &index) const override**
Perform an index operation.
- virtual **GarbageCollected __slice (const GarbageCollected &begin, const GarbageCollected &end, const GarbageCollected &skip) const override**
Perform a slice operation.
- virtual **GarbageCollected __getIterator (const GarbageCollected &collection) const override**
Get an iterator for the expression.
- virtual **GarbageCollected __iteratorNext (size_t index) const override**
Get the next iterative value.
- virtual **GarbageCollected __assign_index (const GarbageCollected &index, const GarbageCollected &value) override**
Perform an index assignment to the supplied value.
- virtual **GarbageCollected __string () const override**

- `const std::vector< Tang::GarbageCollected > & getContents () const`
Return the contents of this object.
- `void append (const Tang::GarbageCollected &item)`
Append an item to the contents of this array object.
- `virtual std::string __asCode () const`
Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- `virtual bool is_equal (const Tang::integer_t &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const Tang::float_t &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const bool &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const std::string &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const Error &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const std::nullptr_t &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual GarbageCollected __add (const GarbageCollected &rhs) const`
Compute the result of adding this value and the supplied value.
- `virtual GarbageCollected __subtract (const GarbageCollected &rhs) const`
Compute the result of subtracting this value and the supplied value.
- `virtual GarbageCollected __multiply (const GarbageCollected &rhs) const`
Compute the result of multiplying this value and the supplied value.
- `virtual GarbageCollected __divide (const GarbageCollected &rhs) const`
Compute the result of dividing this value and the supplied value.
- `virtual GarbageCollected __modulo (const GarbageCollected &rhs) const`
Compute the result of moduloing this value and the supplied value.
- `virtual GarbageCollected __negative () const`
Compute the result of negating this value.
- `virtual GarbageCollected __not () const`
Compute the logical not of this value.
- `virtual GarbageCollected __lessThan (const GarbageCollected &rhs) const`
Compute the "less than" comparison.
- `virtual GarbageCollected __equal (const GarbageCollected &rhs) const`
Perform an equality test.
- `virtual GarbageCollected __period (const GarbageCollected &member, std::shared_ptr< TangBase > &tang) const`
Perform a member access (period) operation.
- `virtual GarbageCollected __integer () const`
Perform a type cast to integer.
- `virtual GarbageCollected __float () const`
Perform a type cast to float.
- `virtual GarbageCollected __boolean () const`
Perform a type cast to boolean.

Static Public Member Functions

- `static NativeBoundFunctionMap getMethods ()`
Return the member functions implemented for this particular expression type.

Private Attributes

- `std::vector< Tang::GarbageCollected > contents`
The array contents.

5.32.1 Detailed Description

Represents an Array that is the result of a computation.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 ComputedExpressionArray()

```
ComputedExpressionArray::ComputedExpressionArray (   
    std::vector< Tang::GarbageCollected > contents )
```

Construct an Array result.

Parameters

<code>val</code>	The integer value.
------------------	--------------------

5.32.3 Member Function Documentation

5.32.3.1 __add()

```
GarbageCollected ComputedExpression::__add (   
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of adding this value and the supplied value.

Parameters

<code>rhs</code>	The <code>GarbageCollected</code> value to add to this.
------------------	---

Returns

The result of the operation.

Reimplemented in `Tang::ComputedExpressionString`, `Tang::ComputedExpressionInteger`, `Tang::ComputedExpressionFloat`, and `Tang::ComputedExpressionError`.

5.32.3.2 `__asCode()`

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.32.3.3 `__assign_index()`

```
GarbageCollected ComputedExpressionArray::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [override], [virtual]
```

Perform an index assignment to the supplied value.

Parameters

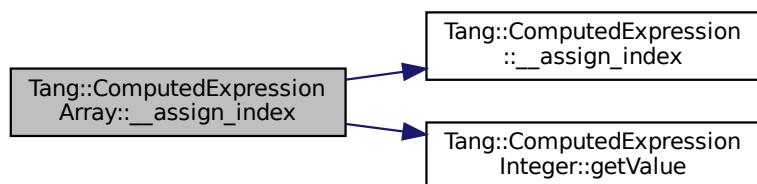
<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.32.3.4 __boolean()

```
GarbageCollected ComputedExpression::__boolean ( ) const [virtual], [inherited]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.32.3.5 __divide()

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.32.3.6 __equal()

```
GarbageCollected ComputedExpression::__equal (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionNativeLibraryFunction](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), and [Tang::ComputedExpressionBoolean](#).

5.32.3.7 `__float()`

`GarbageCollected` `ComputedExpression::__float () const [virtual], [inherited]`

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.32.3.8 `__getIterator()`

`GarbageCollected` `ComputedExpressionArray::__getIterator (`
`const GarbageCollected & collection) const [override], [virtual]`

Get an iterator for the expression.

Parameters

<code>collection</code>	The <code>GarbageCollected</code> value that will serve as the collection through which to iterate.
-------------------------	---

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.9 `__index()`

`GarbageCollected` `ComputedExpressionArray::__index (`
`const GarbageCollected & index) const [override], [virtual]`

Perform an index operation.

Parameters

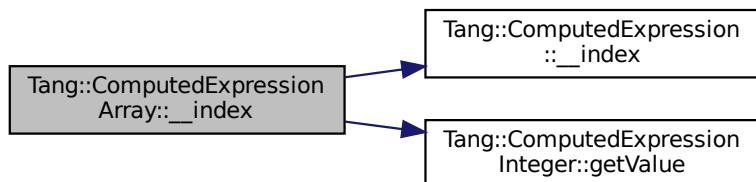
<code>index</code>	The index expression provided by the script.
--------------------	--

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:

**5.32.3.10 __integer()**

[GarbageCollected](#) `ComputedExpression::__integer () const [virtual], [inherited]`

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.32.3.11 __iteratorNext()

[GarbageCollected](#) `ComputedExpressionArray::__iteratorNext (size_t index) const [override], [virtual]`

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.12 `__lessThan()`

```
GarbageCollected ComputedExpression::__lessThan (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to compare against.
------------	---

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.32.3.13 `__modulo()`

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to modulo this by.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.32.3.14 `__multiply()`

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to multiply to this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.32.3.15 `__negative()`

```
GarbageCollected ComputedExpression::__negative ( ) const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.32.3.16 `__not()`

```
GarbageCollected ComputedExpression::__not ( ) const [virtual], [inherited]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.32.3.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<i>member</i>	The member expression provided by the script.
---------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.32.3.18 __slice()

```
GarbageCollected ComputedExpressionArray::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [override], [virtual]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

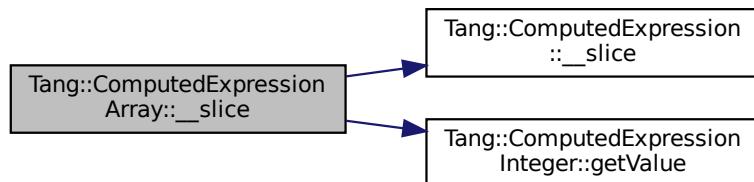
<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.32.3.19 `__string()`

```
GarbageCollected ComputedExpressionArray::__string ( ) const [override], [virtual]
```

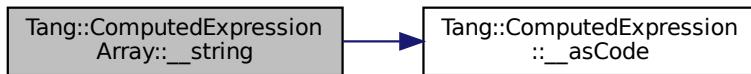
Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.32.3.20 `__subtract()`

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<code>rhs</code>	The GarbageCollected value to subtract from this.
------------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.32.3.21 `append()`

```
void ComputedExpressionArray::append (
    const Tang::GarbageCollected & item )
```

Append an item to the contents of this array object.

Parameters

<i>item</i>	The value to append to the this array.
-------------	--

5.32.3.22 dump()

```
string ComputedExpressionArray::dump ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.23 getContents()

```
const std::vector< Tang::GarbageCollected > & ComputedExpressionArray::getContents ( ) const
```

Return the contents of this object.

Returns

The contents of this object.

5.32.3.24 getMethods()

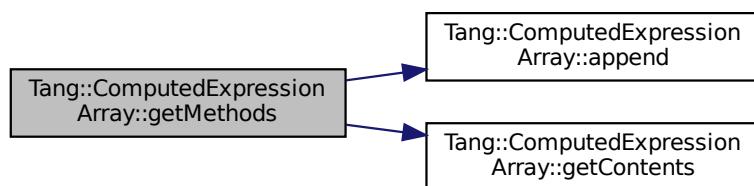
```
NativeBoundFunctionMap ComputedExpressionArray::getMethods ( ) [static]
```

Return the member functions implemented for this particular expression type.

Returns

The member functions implemented.

Here is the call graph for this function:



5.32.3.25 `is_equal()` [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.32.3.26 `is_equal()` [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.32.3.27 `is_equal()` [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

5.32.3.28 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.32.3.29 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.32.3.30 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.32.3.31 isCopyNeeded()

```
bool ComputedExpressionArray::isCopyNeeded ( ) const [override], [virtual]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.32 makeCopy()

```
GarbageCollected ComputedExpressionArray::makeCopy ( ) const [override], [virtual]
```

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented from [Tang::ComputedExpression](#).

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

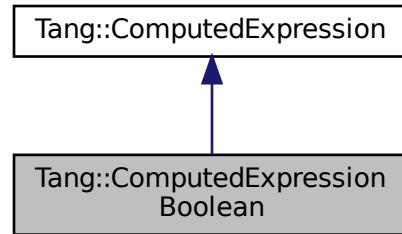
- [include/computedExpressionArray.hpp](#)
- [src/computedExpressionArray.cpp](#)

5.33 Tang::ComputedExpressionBoolean Class Reference

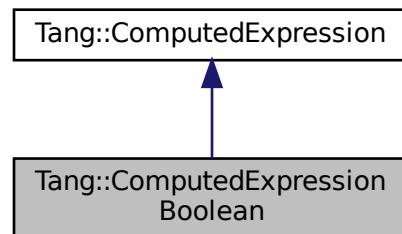
Represents an Boolean that is the result of a computation.

```
#include <computedExpressionBoolean.hpp>
```

Inheritance diagram for Tang::ComputedExpressionBoolean:



Collaboration diagram for Tang::ComputedExpressionBoolean:



Public Member Functions

- [ComputedExpressionBoolean \(bool val\)](#)
Construct an Boolean result.
- [virtual std::string dump \(\) const override](#)
Output the contents of the [ComputedExpression](#) as a string.
- [GarbageCollected makeCopy \(\) const override](#)
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- [virtual bool is_equal \(const bool &val\) const override](#)
Check whether or not the computed expression is equal to another value.
- [virtual GarbageCollected __not \(\) const override](#)
Compute the logical not of this value.

- virtual `GarbageCollected __equal` (const `GarbageCollected &rhs`) const override
Perform an equality test.
- virtual `GarbageCollected __integer` () const override
Perform a type cast to integer.
- virtual `GarbageCollected __float` () const override
Perform a type cast to float.
- virtual `GarbageCollected __boolean` () const override
Perform a type cast to boolean.
- virtual `std::string __asCode` () const
Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- virtual `bool isCopyNeeded` () const
Determine whether or not a copy is needed.
- virtual `bool is_equal` (const `Tang::integer_t &val`) const
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal` (const `Tang::float_t &val`) const
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal` (const `std::string &val`) const
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal` (const `Error &val`) const
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal` (const `std::nullptr_t &val`) const
Check whether or not the computed expression is equal to another value.
- virtual `GarbageCollected __assign_index` (const `GarbageCollected &index`, const `GarbageCollected &value`)
Perform an index assignment to the supplied value.
- virtual `GarbageCollected __add` (const `GarbageCollected &rhs`) const
Compute the result of adding this value and the supplied value.
- virtual `GarbageCollected __subtract` (const `GarbageCollected &rhs`) const
Compute the result of subtracting this value and the supplied value.
- virtual `GarbageCollected __multiply` (const `GarbageCollected &rhs`) const
Compute the result of multiplying this value and the supplied value.
- virtual `GarbageCollected __divide` (const `GarbageCollected &rhs`) const
Compute the result of dividing this value and the supplied value.
- virtual `GarbageCollected __modulo` (const `GarbageCollected &rhs`) const
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __negative` () const
Compute the result of negating this value.
- virtual `GarbageCollected __lessThan` (const `GarbageCollected &rhs`) const
Compute the "less than" comparison.
- virtual `GarbageCollected __period` (const `GarbageCollected &member`, `std::shared_ptr<TangBase> &tang`) const
Perform a member access (period) operation.
- virtual `GarbageCollected __index` (const `GarbageCollected &index`) const
Perform an index operation.
- virtual `GarbageCollected __slice` (const `GarbageCollected &begin`, const `GarbageCollected &end`, const `GarbageCollected &skip`) const
Perform a slice operation.
- virtual `GarbageCollected __getIterator` (const `GarbageCollected &collection`) const
Get an iterator for the expression.
- virtual `GarbageCollected __iteratorNext` (`size_t index=0`) const
Get the next iterative value.
- virtual `GarbageCollected __string` () const
Perform a type cast to string.

Private Attributes

- bool `val`
The boolean value.

5.33.1 Detailed Description

Represents an Boolean that is the result of a computation.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 ComputedExpressionBoolean()

```
ComputedExpressionBoolean::ComputedExpressionBoolean (   
    bool val )
```

Construct an Boolean result.

Parameters

<code>val</code>	The boolean value.
------------------	--------------------

5.33.3 Member Function Documentation

5.33.3.1 __add()

```
GarbageCollected ComputedExpression::__add (   
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of adding this value and the supplied value.

Parameters

<code>rhs</code>	The <code>GarbageCollected</code> value to add to this.
------------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.33.3.2 `__asCode()`

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.33.3.3 `__assign_index()`

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.33.3.4 `__boolean()`

```
GarbageCollected ComputedExpressionBoolean::__boolean ( ) const [override], [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.33.3.5 `__divide()`

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.33.3.6 `__equal()`

```
GarbageCollected ComputedExpressionBoolean::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:

**5.33.3.7 `__float()`**

```
GarbageCollected ComputedExpressionBoolean::__float ( ) const [override], [virtual]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.33.3.8 `__getIterator()`

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The <code>GarbageCollected</code> value that will serve as the collection through which to iterate.
-------------------	---

Reimplemented in `Tang::ComputedExpressionString`, `Tang::ComputedExpressionMap`, and `Tang::ComputedExpressionArray`.

5.33.3.9 `__index()`

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in `Tang::ComputedExpressionString`, `Tang::ComputedExpressionMap`, and `Tang::ComputedExpressionArray`.

5.33.3.10 `__integer()`

```
GarbageCollected ComputedExpressionBoolean::__integer ( ) const [override], [virtual]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented from `Tang::ComputedExpression`.

5.33.3.11 `__iteratorNext()`

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.33.3.12 __lessThan()

```
GarbageCollected ComputedExpression::__lessThan (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.33.3.13 __modulo()

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.33.3.14 `__multiply()`

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<code>rhs</code>	The <code>GarbageCollected</code> value to multiply to this.
------------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.33.3.15 `__negative()`

```
GarbageCollected ComputedExpression::__negative ( ) const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.33.3.16 `__not()`

```
GarbageCollected ComputedExpressionBoolean::__not ( ) const [override], [virtual]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.33.3.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<i>member</i>	The member expression provided by the script.
---------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.33.3.18 __slice()

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.33.3.19 __string()

```
GarbageCollected ComputedExpression::__string ( ) const [virtual], [inherited]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.33.3.20 `__subtract()`

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to subtract from this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.33.3.21 `dump()`

```
string ComputedExpressionBoolean::dump ( ) const [override], [virtual]
```

Output the contents of the `ComputedExpression` as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.33.3.22 `is_equal()` [1/6]

```
bool ComputedExpressionBoolean::is_equal (
    const bool & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.33.3.23 `is_equal()` [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.33.3.24 `is_equal()` [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

5.33.3.25 `is_equal()` [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.33.3.26 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.33.3.27 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.33.3.28 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.33.3.29 makeCopy()

`GarbageCollected` [ComputedExpressionBoolean::makeCopy](#) () const [override], [virtual]

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented from [Tang::ComputedExpression](#).

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

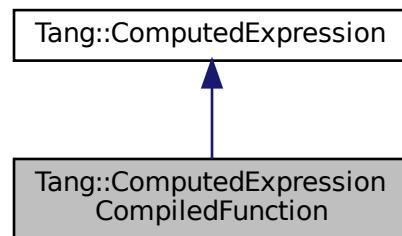
- [include/computedExpressionBoolean.hpp](#)
- [src/computedExpressionBoolean.cpp](#)

5.34 Tang::ComputedExpressionCompiledFunction Class Reference

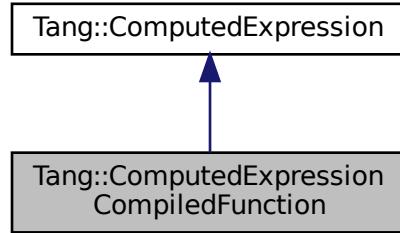
Represents a Compiled Function declared in the script.

```
#include <computedExpressionCompiledFunction.hpp>
```

Inheritance diagram for Tang::ComputedExpressionCompiledFunction:



Collaboration diagram for Tang::ComputedExpressionCompiledFunction:



Public Member Functions

- **ComputedExpressionCompiledFunction** (uint32_t argc, Tang::integer_t pc)
Construct an CompiledFunction.
- virtual std::string **dump** () const override
Output the contents of the [ComputedExpression](#) as a string.
- **GarbageCollected makeCopy** () const override
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual **GarbageCollected __equal** (const [GarbageCollected](#) &rhs) const override
Perform an equality test.
- uint32_t **getArgc** () const
Get the argc value.
- Tang::integer_t **getPc** () const
Get the bytecode target.
- virtual std::string **__asCode** () const
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- virtual bool **isCopyNeeded** () const
Determine whether or not a copy is needed.
- virtual bool **is_equal** (const Tang::integer_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool **is_equal** (const Tang::float_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool **is_equal** (const bool &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool **is_equal** (const std::string &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool **is_equal** (const [Error](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool **is_equal** (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual **GarbageCollected __assign_index** (const [GarbageCollected](#) &index, const [GarbageCollected](#) &value)
Perform an index assignment to the supplied value.
- virtual **GarbageCollected __add** (const [GarbageCollected](#) &rhs) const
Compute the result of adding this value and the supplied value.

- virtual `GarbageCollected __subtract` (const `GarbageCollected` &rhs) const
Compute the result of subtracting this value and the supplied value.
- virtual `GarbageCollected __multiply` (const `GarbageCollected` &rhs) const
Compute the result of multiplying this value and the supplied value.
- virtual `GarbageCollected __divide` (const `GarbageCollected` &rhs) const
Compute the result of dividing this value and the supplied value.
- virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __negative` () const
Compute the result of negating this value.
- virtual `GarbageCollected __not` () const
Compute the logical not of this value.
- virtual `GarbageCollected __lessThan` (const `GarbageCollected` &rhs) const
Compute the "less than" comparison.
- virtual `GarbageCollected __period` (const `GarbageCollected` &member, std::shared_ptr<`TangBase`> &tang) const
Perform a member access (period) operation.
- virtual `GarbageCollected __index` (const `GarbageCollected` &index) const
Perform an index operation.
- virtual `GarbageCollected __slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const
Perform a slice operation.
- virtual `GarbageCollected __getIterator` (const `GarbageCollected` &collection) const
Get an iterator for the expression.
- virtual `GarbageCollected __iteratorNext` (size_t index=0) const
Get the next iterative value.
- virtual `GarbageCollected __integer` () const
Perform a type cast to integer.
- virtual `GarbageCollected __float` () const
Perform a type cast to float.
- virtual `GarbageCollected __boolean` () const
Perform a type cast to boolean.
- virtual `GarbageCollected __string` () const
Perform a type cast to string.

Private Attributes

- `uint32_t argc`
The count of arguments that this function expects.
- `Tang::integer_t pc`
The bytecode address of the start of the function.

5.34.1 Detailed Description

Represents a Compiled Function declared in the script.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 ComputedExpressionCompiledFunction()

```
ComputedExpressionCompiledFunction::ComputedExpressionCompiledFunction (
    uint32_t argc,
    Tang::integer_t pc )
```

Construct an CompiledFunction.

Parameters

<i>argc</i>	The count of arguments that this function expects.
<i>pc</i>	The bytecode address of the start of the function.

5.34.3 Member Function Documentation

5.34.3.1 __add()

```
GarbageCollected ComputedExpression::__add (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.34.3.2 __asCode()

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.34.3.3 `__assign_index()`

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value )  [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.34.3.4 `__boolean()`

```
GarbageCollected ComputedExpression::__boolean () const [virtual], [inherited]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.34.3.5 `__divide()`

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.34.3.6 __equal()

```
GarbageCollected ComputedExpressionCompiledFunction::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:

**5.34.3.7 __float()**

```
GarbageCollected ComputedExpression::__float () const [virtual], [inherited]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.34.3.8 __getIterator()

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.34.3.9 __index()

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.34.3.10 __integer()

```
GarbageCollected ComputedExpression::__integer ( ) const [virtual], [inherited]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.34.3.11 __iteratorNext()

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.34.3.12 __lessThan()

```
GarbageCollected ComputedExpression::__lessThan (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.34.3.13 __modulo()

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.34.3.14 `__multiply()`

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to multiply to this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.34.3.15 `__negative()`

```
GarbageCollected ComputedExpression::__negative () const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.34.3.16 `__not()`

```
GarbageCollected ComputedExpression::__not () const [virtual], [inherited]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.34.3.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<i>member</i>	The member expression provided by the script.
---------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.34.3.18 `__slice()`

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.34.3.19 `__string()`

```
GarbageCollected ComputedExpression::__string ( ) const [virtual], [inherited]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.34.3.20 `__subtract()`

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to subtract from this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.34.3.21 `dump()`

```
string ComputedExpressionCompiledFunction::dump ( ) const [override], [virtual]
```

Output the contents of the `ComputedExpression` as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.34.3.22 `is_equal()` [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.34.3.23 `is_equal()` [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.34.3.24 `is_equal()` [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

5.34.3.25 `is_equal()` [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.34.3.26 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.34.3.27 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.34.3.28 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for `ComputedExpressions` which serve as containers, such as `ComputedExpressionArray` and `ComputedExpressionObject`.

Returns

Whether or not a copy is needed.

Reimplemented in `Tang::ComputedExpressionMap`, and `Tang::ComputedExpressionArray`.

5.34.3.29 `makeCopy()`

`GarbageCollected` `ComputedExpressionCompiledFunction::makeCopy () const [override], [virtual]`

Make a copy of the `ComputedExpression` (recursively, if appropriate).

Returns

A `Tang::GarbageCollected` value for the new `ComputedExpression`.

Reimplemented from `Tang::ComputedExpression`.

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

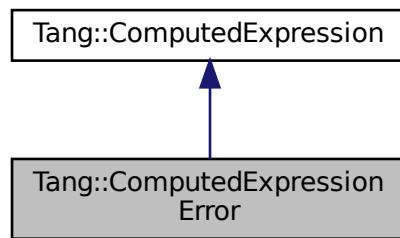
- `include/computedExpressionCompiledFunction.hpp`
- `src/computedExpressionCompiledFunction.cpp`

5.35 `Tang::ComputedExpressionError` Class Reference

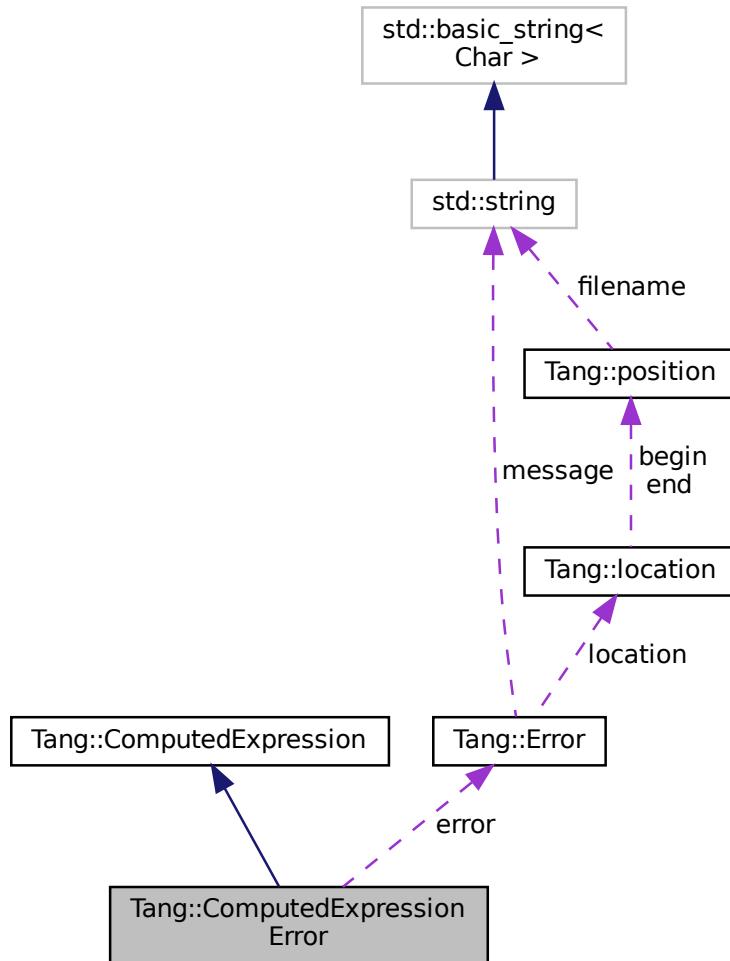
Represents a Runtime `Error`.

```
#include <computedExpressionError.hpp>
```

Inheritance diagram for `Tang::ComputedExpressionError`:



Collaboration diagram for Tang::ComputedExpressionError:



Public Member Functions

- `ComputedExpressionError (Tang::Error error)`
Construct a Runtime Error.
- `virtual std::string dump () const override`
Output the contents of the `ComputedExpression` as a string.
- `GarbageCollected makeCopy () const override`
Make a copy of the `ComputedExpression` (recursively, if appropriate).
- `virtual bool is_equal (const Error &val) const override`
Check whether or not the computed expression is equal to another value.
- `virtual GarbageCollected __add (const GarbageCollected &rhs) const override`
Compute the result of adding this value and the supplied value.
- `virtual GarbageCollected __subtract (const GarbageCollected &rhs) const override`
Compute the result of subtracting this value and the supplied value.
- `virtual GarbageCollected __multiply (const GarbageCollected &rhs) const override`

- virtual `GarbageCollected __divide` (const `GarbageCollected &rhs`) const override

Compute the result of dividing this value and the supplied value.
- virtual `GarbageCollected __modulo` (const `GarbageCollected &rhs`) const override

Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __negative` () const override

Compute the result of negating this value.
- virtual `GarbageCollected __not` () const override

Compute the logical not of this value.
- virtual `GarbageCollected __lessThan` (const `GarbageCollected &rhs`) const override

Compute the "less than" comparison.
- virtual `GarbageCollected __equal` (const `GarbageCollected &rhs`) const override

Perform an equality test.
- virtual `GarbageCollected __integer` () const override

Perform a type cast to integer.
- virtual `GarbageCollected __float` () const override

Perform a type cast to float.
- virtual `GarbageCollected __boolean` () const override

Perform a type cast to boolean.
- virtual `GarbageCollected __string` () const override

Perform a type cast to string.
- virtual `std::string __asCode` () const

Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- virtual `bool isCopyNeeded` () const

Determine whether or not a copy is needed.
- virtual `bool is_equal` (const `Tang::integer_t &val`) const

Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal` (const `Tang::float_t &val`) const

Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal` (const `bool &val`) const

Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal` (const `std::string &val`) const

Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal` (const `std::nullptr_t &val`) const

Check whether or not the computed expression is equal to another value.
- virtual `GarbageCollected __assign_index` (const `GarbageCollected &index`, const `GarbageCollected &value`)

Perform an index assignment to the supplied value.
- virtual `GarbageCollected __period` (const `GarbageCollected &member`, `std::shared_ptr<TangBase> &tang`) const

Perform a member access (period) operation.
- virtual `GarbageCollected __index` (const `GarbageCollected &index`) const

Perform an index operation.
- virtual `GarbageCollected __slice` (const `GarbageCollected &begin`, const `GarbageCollected &end`, const `GarbageCollected &skip`) const

Perform a slice operation.
- virtual `GarbageCollected __getIterator` (const `GarbageCollected &collection`) const

Get an iterator for the expression.
- virtual `GarbageCollected __iteratorNext` (`size_t index=0`) const

Get the next iterative value.

Private Attributes

- `Tang::Error error`

The `Error` object.

5.35.1 Detailed Description

Represents a Runtime [Error](#).

5.35.2 Constructor & Destructor Documentation

5.35.2.1 ComputedExpressionError()

```
ComputedExpressionError::ComputedExpressionError (
```

<code>Tang::Error</code>	<code>error</code>
--------------------------	--------------------

```
        )
```

Construct a Runtime [Error](#).

Parameters

<code>error</code>	The <code>Tang::Error</code> object.
--------------------	--------------------------------------

5.35.3 Member Function Documentation

5.35.3.1 __add()

```
GarbageCollected ComputedExpressionError::__add (
```

<code>const GarbageCollected</code>	<code>& rhs</code>
-------------------------------------	------------------------

```
        ) const [override], [virtual]
```

Compute the result of adding this value and the supplied value.

Parameters

<code>rhs</code>	The <code>GarbageCollected</code> value to add to this.
------------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.2 `__asCode()`

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.35.3.3 `__assign_index()`

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.35.3.4 `__boolean()`

```
GarbageCollected ComputedExpressionError::__boolean ( ) const [override], [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.5 `__divide()`

```
GarbageCollected ComputedExpressionError::__divide (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.6 [__equal\(\)](#)

```
GarbageCollected ComputedExpressionError::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.7 [__float\(\)](#)

```
GarbageCollected ComputedExpressionError::__float ( ) const [override], [virtual]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.8 [__getIterator\(\)](#)

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.35.3.9 __index()

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.35.3.10 __integer()

```
GarbageCollected ComputedExpressionError::__integer ( ) const [override], [virtual]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.11 __iteratorNext()

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.35.3.12 `__lessThan()`

```
GarbageCollected ComputedExpressionError::__lessThan (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.13 `__modulo()`

```
GarbageCollected ComputedExpressionError::__modulo (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.14 `__multiply()`

```
GarbageCollected ComputedExpressionError::__multiply (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<code>rhs</code>	The <code>GarbageCollected</code> value to multiply to this.
------------------	--

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.15 `__negative()`

```
GarbageCollected ComputedExpressionError::__negative ( ) const [override], [virtual]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.16 `__not()`

```
GarbageCollected ComputedExpressionError::__not ( ) const [override], [virtual]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<i>member</i>	The member expression provided by the script.
---------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.35.3.18 `__slice()`

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.35.3.19 `__string()`

```
GarbageCollected ComputedExpressionError::__string ( ) const [override], [virtual]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.20 `__subtract()`

```
GarbageCollected ComputedExpressionError::__subtract (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to subtract from this.
------------	--

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.21 `dump()`

```
std::string ComputedExpressionError::dump ( ) const [override], [virtual]
```

Output the contents of the `ComputedExpression` as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.22 `is_equal()` [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.35.3.23 is_equal() [2/6]

```
bool ComputedExpressionError::is_equal (
    const Error & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

val	The value to compare against.
-----	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.35.3.24 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

val	The value to compare against.
-----	-------------------------------

Returns

True if equal, false if not.

5.35.3.25 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

val	The value to compare against.
-----	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.35.3.26 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.35.3.27 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.35.3.28 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for **ComputedExpressions** which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.35.3.29 **makeCopy()**

[GarbageCollected](#) `ComputedExpressionError::makeCopy () const [override], [virtual]`

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented from [Tang::ComputedExpression](#).

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

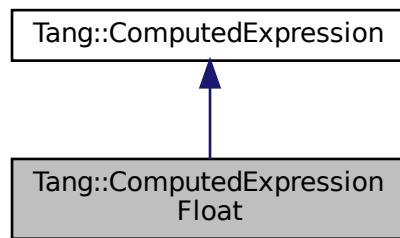
- [include/computedExpressionError.hpp](#)
- [src/computedExpressionError.cpp](#)

5.36 **Tang::ComputedExpressionFloat** Class Reference

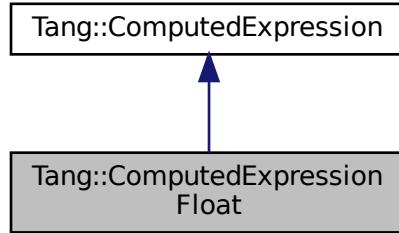
Represents a **Float** that is the result of a computation.

```
#include <computedExpressionFloat.hpp>
```

Inheritance diagram for [Tang::ComputedExpressionFloat](#):



Collaboration diagram for Tang::ComputedExpressionFloat:



Public Member Functions

- **ComputedExpressionFloat (Tang::float_t val)**
Construct a Float result.
- virtual std::string **dump () const override**
Output the contents of the [ComputedExpression](#) as a string.
- **GarbageCollected makeCopy () const override**
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual bool **is_equal (const Tang::integer_t &val) const override**
Check whether or not the computed expression is equal to another value.
- virtual bool **is_equal (const Tang::float_t &val) const override**
Check whether or not the computed expression is equal to another value.
- virtual bool **is_equal (const bool &val) const override**
Check whether or not the computed expression is equal to another value.
- virtual **GarbageCollected __add (const GarbageCollected &rhs) const override**
Compute the result of adding this value and the supplied value.
- virtual **GarbageCollected __subtract (const GarbageCollected &rhs) const override**
Compute the result of subtracting this value and the supplied value.
- virtual **GarbageCollected __multiply (const GarbageCollected &rhs) const override**
Compute the result of multiplying this value and the supplied value.
- virtual **GarbageCollected __divide (const GarbageCollected &rhs) const override**
Compute the result of dividing this value and the supplied value.
- virtual **GarbageCollected __negative () const override**
Compute the result of negating this value.
- virtual **GarbageCollected __not () const override**
Compute the logical not of this value.
- virtual **GarbageCollected __lessThan (const GarbageCollected &rhs) const override**
Compute the "less than" comparison.
- virtual **GarbageCollected __equal (const GarbageCollected &rhs) const override**
Perform an equality test.
- virtual **GarbageCollected __integer () const override**
Perform a type cast to integer.
- virtual **GarbageCollected __float () const override**
Perform a type cast to float.

- virtual `GarbageCollected __boolean () const override`
Perform a type cast to boolean.
- virtual `GarbageCollected __string () const override`
Perform a type cast to string.
- `Tang::float_t getValue () const`
Helper function to get the value associated with this expression.
- virtual `std::string __asCode () const`
Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- virtual `bool isCopyNeeded () const`
Determine whether or not a copy is needed.
- virtual `bool is_equal (const std::string &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal (const Error &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal (const std::nullptr_t &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `GarbageCollected __assign_index (const GarbageCollected &index, const GarbageCollected &value)`
Perform an index assignment to the supplied value.
- virtual `GarbageCollected __modulo (const GarbageCollected &rhs) const`
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __period (const GarbageCollected &member, std::shared_ptr< TangBase > &tang) const`
Perform a member access (period) operation.
- virtual `GarbageCollected __index (const GarbageCollected &index) const`
Perform an index operation.
- virtual `GarbageCollected __slice (const GarbageCollected &begin, const GarbageCollected &end, const GarbageCollected &skip) const`
Perform a slice operation.
- virtual `GarbageCollected __getIterator (const GarbageCollected &collection) const`
Get an iterator for the expression.
- virtual `GarbageCollected __iteratorNext (size_t index=0) const`
Get the next iterative value.

Private Attributes

- `Tang::float_t val`
The float value.

5.36.1 Detailed Description

Represents a Float that is the result of a computation.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 ComputedExpressionFloat()

```
ComputedExpressionFloat::ComputedExpressionFloat (
    Tang::float_t val )
```

Construct a Float result.

Parameters

<i>val</i>	The float value.
------------	------------------

5.36.3 Member Function Documentation

5.36.3.1 `__add()`

```
GarbageCollected ComputedExpressionFloat::__add (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of adding this value and the supplied value.

Parameters

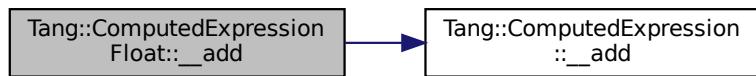
<i>rhs</i>	The <code>GarbageCollected</code> value to add to this.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.36.3.2 `__asCode()`

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.36.3.3 `__assign_index()`

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value )  [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.36.3.4 `__boolean()`

```
GarbageCollected ComputedExpressionFloat::__boolean ( ) const [override], [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.36.3.5 `__divide()`

```
GarbageCollected ComputedExpressionFloat::__divide (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of dividing this value and the supplied value.

Parameters

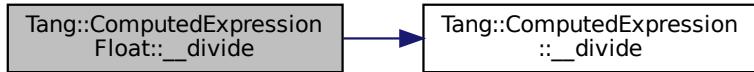
<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.36.3.6 __equal()

```
GarbageCollected ComputedExpressionFloat::__equal (\n    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.36.3.7 __float()

```
GarbageCollected ComputedExpressionFloat::__float () const [override], [virtual]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.36.3.8 `__getIterator()`

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.36.3.9 `__index()`

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.36.3.10 `__integer()`

```
GarbageCollected ComputedExpressionFloat::__integer ( ) const [override], [virtual]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.36.3.11 `__iteratorNext()`

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.36.3.12 `__lessThan()`

```
GarbageCollected ComputedExpressionFloat::__lessThan (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the "less than" comparison.

Parameters

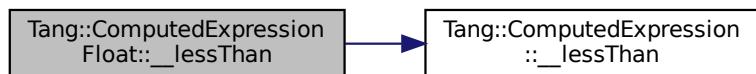
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.36.3.13 `__modulo()`

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.36.3.14 [__multiply\(\)](#)

```
GarbageCollected ComputedExpressionFloat::__multiply (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.36.3.15 [__negative\(\)](#)

```
GarbageCollected ComputedExpressionFloat::__negative ( ) const [override], [virtual]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.36.3.16 `__not()`

```
GarbageCollected ComputedExpressionFloat::__not ( ) const [override], [virtual]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.36.3.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<i>member</i>	The member expression provided by the script.
---------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.36.3.18 `__slice()`

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.36.3.19 `__string()`

```
GarbageCollected ComputedExpressionFloat::__string ( ) const [override], [virtual]
```

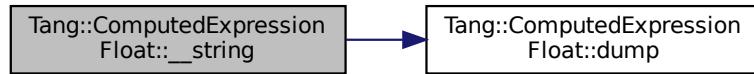
Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.36.3.20 `__subtract()`

```
GarbageCollected ComputedExpressionFloat::__subtract ( \n    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of subtracting this value and the supplied value.

Parameters

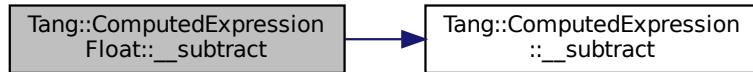
<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.36.3.21 `dump()`

```
string ComputedExpressionFloat::dump ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.36.3.22 `getValue()`

```
Tang::float_t ComputedExpressionFloat::getValue ( ) const
```

Helper function to get the value associated with this expression.

Returns

The value associated with this expression.

5.36.3.23 `is_equal()` [1/6]

```
bool ComputedExpressionFloat::is_equal ( & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.36.3.24 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.36.3.25 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

5.36.3.26 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.36.3.27 is_equal() [5/6]

```
bool ComputedExpressionFloat::is_equal (
    const Tang::float_t & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.36.3.28 is_equal() [6/6]

```
bool ComputedExpressionFloat::is_equal (
    const Tang::integer_t & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.36.3.29 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.36.3.30 makeCopy()

```
GarbageCollected ComputedExpressionFloat::makeCopy ( ) const [override], [virtual]
```

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented from [Tang::ComputedExpression](#).

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

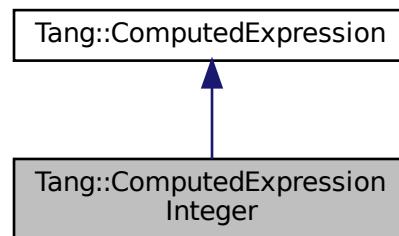
- [include/computedExpressionFloat.hpp](#)
- [src/computedExpressionFloat.cpp](#)

5.37 Tang::ComputedExpressionInteger Class Reference

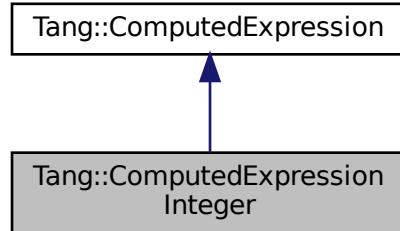
Represents an Integer that is the result of a computation.

```
#include <computedExpressionInteger.hpp>
```

Inheritance diagram for Tang::ComputedExpressionInteger:



Collaboration diagram for Tang::ComputedExpressionInteger:



Public Member Functions

- **ComputedExpressionInteger (Tang::integer_t val)**
Construct an Integer result.
- virtual std::string **dump () const override**
Output the contents of the [ComputedExpression](#) as a string.
- **GarbageCollected makeCopy () const override**
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual bool **is_equal (const Tang::integer_t &val) const override**
Check whether or not the computed expression is equal to another value.
- virtual bool **is_equal (const Tang::float_t &val) const override**
Check whether or not the computed expression is equal to another value.
- virtual bool **is_equal (const bool &val) const override**
Check whether or not the computed expression is equal to another value.
- virtual **GarbageCollected __add (const GarbageCollected &rhs) const override**
Compute the result of adding this value and the supplied value.
- virtual **GarbageCollected __subtract (const GarbageCollected &rhs) const override**
Compute the result of subtracting this value and the supplied value.
- virtual **GarbageCollected __multiply (const GarbageCollected &rhs) const override**
Compute the result of multiplying this value and the supplied value.
- virtual **GarbageCollected __divide (const GarbageCollected &rhs) const override**
Compute the result of dividing this value and the supplied value.
- virtual **GarbageCollected __modulo (const GarbageCollected &rhs) const override**
Compute the result of moduloing this value and the supplied value.
- virtual **GarbageCollected __negative () const override**
Compute the result of negating this value.
- virtual **GarbageCollected __not () const override**
Compute the logical not of this value.
- virtual **GarbageCollected __lessThan (const GarbageCollected &rhs) const override**
Compute the "less than" comparison.
- virtual **GarbageCollected __equal (const GarbageCollected &rhs) const override**
Perform an equality test.
- virtual **GarbageCollected __integer () const override**
Perform a type cast to integer.

- virtual `GarbageCollected __float () const` override
Perform a type cast to float.
- virtual `GarbageCollected __boolean () const` override
Perform a type cast to boolean.
- virtual `GarbageCollected __string () const` override
Perform a type cast to string.
- `Tang::integer_t getValue () const`
Helper function to get the value associated with this expression.
- `virtual std::string __asCode () const`
Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- `virtual bool isCopyNeeded () const`
Determine whether or not a copy is needed.
- `virtual bool is_equal (const std::string &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const Error &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const std::nullptr_t &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `GarbageCollected __assign_index (const GarbageCollected &index, const GarbageCollected &value)`
Perform an index assignment to the supplied value.
- `virtual GarbageCollected __period (const GarbageCollected &member, std::shared_ptr< TangBase > &tang) const`
Perform a member access (period) operation.
- `virtual GarbageCollected __index (const GarbageCollected &index) const`
Perform an index operation.
- `virtual GarbageCollected __slice (const GarbageCollected &begin, const GarbageCollected &end, const GarbageCollected &skip) const`
Perform a slice operation.
- `virtual GarbageCollected __getIterator (const GarbageCollected &collection) const`
Get an iterator for the expression.
- `virtual GarbageCollected __iteratorNext (size_t index=0) const`
Get the next iterative value.

Private Attributes

- `Tang::integer_t val`
The integer value.

5.37.1 Detailed Description

Represents an Integer that is the result of a computation.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 ComputedExpressionInteger()

```
ComputedExpressionInteger::ComputedExpressionInteger (
    Tang::integer_t val )
```

Construct an Integer result.

Parameters

<i>val</i>	The integer value.
------------	--------------------

5.37.3 Member Function Documentation

5.37.3.1 `__add()`

```
GarbageCollected ComputedExpressionInteger::__add (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of adding this value and the supplied value.

Parameters

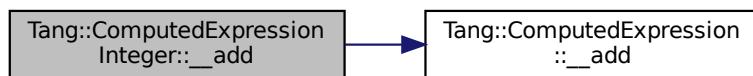
<i>rhs</i>	The <code>GarbageCollected</code> value to add to this.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.37.3.2 `__asCode()`

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.37.3.3 `__assign_index()`

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value )  [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<code>index</code>	The index to which the value should be applied.
<code>value</code>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.37.3.4 `__boolean()`

```
GarbageCollected ComputedExpressionInteger::__boolean ( ) const [override], [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.37.3.5 `__divide()`

```
GarbageCollected ComputedExpressionInteger::__divide (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of dividing this value and the supplied value.

Parameters

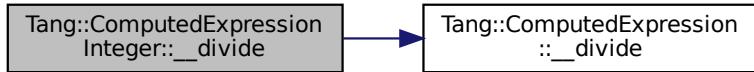
<code>rhs</code>	The GarbageCollected value to divide this by.
------------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.37.3.6 __equal()

```
GarbageCollected ComputedExpressionInteger::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

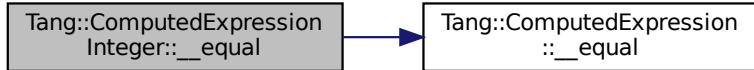
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.37.3.7 __float()

```
GarbageCollected ComputedExpressionInteger::__float () const [override], [virtual]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.37.3.8 `__getIterator()`

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.37.3.9 `__index()`

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.37.3.10 `__integer()`

```
GarbageCollected ComputedExpressionInteger::__integer ( ) const [override], [virtual]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.37.3.11 `__iteratorNext()`

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.37.3.12 `__lessThan()`

```
GarbageCollected ComputedExpressionInteger::__lessThan (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.37.3.13 `__modulo()`

```
GarbageCollected ComputedExpressionInteger::__modulo (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of moduloing this value and the supplied value.

Parameters

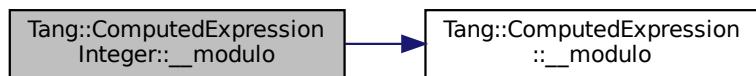
<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.37.3.14 __multiply()

```
GarbageCollected ComputedExpressionInteger::__multiply (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of multiplying this value and the supplied value.

Parameters

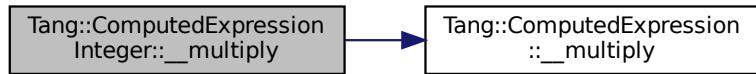
<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.37.3.15 `__negative()`

```
GarbageCollected ComputedExpressionInteger::__negative ( ) const [override], [virtual]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.37.3.16 `__not()`

```
GarbageCollected ComputedExpressionInteger::__not ( ) const [override], [virtual]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.37.3.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<code>member</code>	The member expression provided by the script.
---------------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.37.3.18 `__slice()`

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.37.3.19 `__string()`

```
GarbageCollected ComputedExpressionInteger::__string () const [override], [virtual]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.37.3.20 `__subtract()`

```
GarbageCollected ComputedExpressionInteger::__subtract (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:

**5.37.3.21 dump()**

```
string ComputedExpressionInteger::dump ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.37.3.22 getValue()

```
Tang::integer_t ComputedExpressionInteger::getValue ( ) const
```

Helper function to get the value associated with this expression.

Returns

The value associated with this expression.

5.37.3.23 is_equal() [1/6]

```
bool ComputedExpressionInteger::is_equal (
    const bool & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.37.3.24 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.37.3.25 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.37.3.26 `is_equal()` [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.37.3.27 `is_equal()` [5/6]

```
bool ComputedExpressionInteger::is_equal (
    const Tang::float_t & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.37.3.28 `is_equal()` [6/6]

```
bool ComputedExpressionInteger::is_equal (
    const Tang::integer_t & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.37.3.29 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.37.3.30 makeCopy()

```
GarbageCollected ComputedExpressionInteger::makeCopy ( ) const [override], [virtual]
```

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented from [Tang::ComputedExpression](#).

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

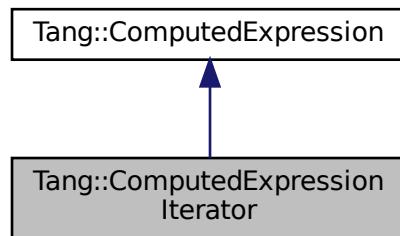
- include/computedExpressionInteger.hpp
- src/computedExpressionInteger.cpp

5.38 Tang::ComputedExpressionIterator Class Reference

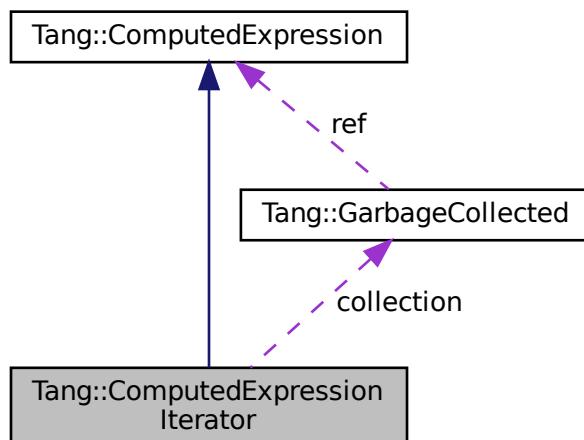
Represents an iterator that is the result of a computation.

```
#include <computedExpressionIterator.hpp>
```

Inheritance diagram for Tang::ComputedExpressionIterator:



Collaboration diagram for Tang::ComputedExpressionIterator:



Public Member Functions

- [ComputedExpressionIterator \(Tang::GarbageCollected collection\)](#)
Construct an iterator result.
- [virtual std::string dump \(\) const override](#)
Output the contents of the [ComputedExpression](#) as a string.

- virtual `GarbageCollected __iteratorNext (size_t index) const` override
Get the next iterative value.
- virtual `std::string __asCode () const`
Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- virtual `bool isCopyNeeded () const`
Determine whether or not a copy is needed.
- virtual `GarbageCollected makeCopy () const`
Make a copy of the `ComputedExpression` (recursively, if appropriate).
- virtual `bool is_equal (const Tang::integer_t &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal (const Tang::float_t &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal (const bool &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal (const std::string &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal (const Error &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal (const std::nullptr_t &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `GarbageCollected __assign_index (const GarbageCollected &index, const GarbageCollected &value)`
Perform an index assignment to the supplied value.
- virtual `GarbageCollected __add (const GarbageCollected &rhs) const`
Compute the result of adding this value and the supplied value.
- virtual `GarbageCollected __subtract (const GarbageCollected &rhs) const`
Compute the result of subtracting this value and the supplied value.
- virtual `GarbageCollected __multiply (const GarbageCollected &rhs) const`
Compute the result of multiplying this value and the supplied value.
- virtual `GarbageCollected __divide (const GarbageCollected &rhs) const`
Compute the result of dividing this value and the supplied value.
- virtual `GarbageCollected __modulo (const GarbageCollected &rhs) const`
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __negative () const`
Compute the result of negating this value.
- virtual `GarbageCollected __not () const`
Compute the logical not of this value.
- virtual `GarbageCollected __lessThan (const GarbageCollected &rhs) const`
Compute the "less than" comparison.
- virtual `GarbageCollected __equal (const GarbageCollected &rhs) const`
Perform an equality test.
- virtual `GarbageCollected __period (const GarbageCollected &member, std::shared_ptr< TangBase > &tang) const`
Perform a member access (period) operation.
- virtual `GarbageCollected __index (const GarbageCollected &index) const`
Perform an index operation.
- virtual `GarbageCollected __slice (const GarbageCollected &begin, const GarbageCollected &end, const GarbageCollected &skip) const`
Perform a slice operation.
- virtual `GarbageCollected __getIterator (const GarbageCollected &collection) const`
Get an iterator for the expression.
- virtual `GarbageCollected __integer () const`

- virtual `GarbageCollected __float () const`

Perform a type cast to float.
- virtual `GarbageCollected __boolean () const`

Perform a type cast to boolean.
- virtual `GarbageCollected __string () const`

Perform a type cast to string.

Private Attributes

- `Tang::GarbageCollected collection`

The target collection.
- `size_t index`

The next index.

5.38.1 Detailed Description

Represents an Iterator that is the result of a computation.

5.38.2 Constructor & Destructor Documentation

5.38.2.1 `ComputedExpressionIterator()`

```
ComputedExpressionIterator::ComputedExpressionIterator (
    Tang::GarbageCollected collection )
```

Construct an Iterator result.

Parameters

<code>collection</code>	The collection through which the iterator processes
-------------------------	---

5.38.3 Member Function Documentation

5.38.3.1 `__add()`

```
GarbageCollected ComputedExpression::__add (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.38.3.2 [__asCode\(\)](#)

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.38.3.3 [__assign_index\(\)](#)

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.38.3.4 __boolean()

```
GarbageCollected ComputedExpression::__boolean ( ) const [virtual], [inherited]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.38.3.5 __divide()

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.38.3.6 __equal()

```
GarbageCollected ComputedExpression::__equal (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionNativeLibraryFunction](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), and [Tang::ComputedExpressionBoolean](#).

5.38.3.7 `__float()`

```
GarbageCollected ComputedExpression::__float () const [virtual], [inherited]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.38.3.8 `__getIterator()`

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<code>collection</code>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------------	--

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.38.3.9 `__index()`

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

<code>index</code>	The index expression provided by the script.
--------------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.38.3.10 `__integer()`

```
GarbageCollected ComputedExpression::__integer ( ) const [virtual], [inherited]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.38.3.11 `__iteratorNext()`

```
GarbageCollected ComputedExpressionIterator::__iteratorNext ( size_t index ) const [override], [virtual]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.38.3.12 `__lessThan()`

```
GarbageCollected ComputedExpression::__lessThan (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to compare against.
------------	---

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.38.3.13 `__modulo()`

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to modulo this by.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.38.3.14 `__multiply()`

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to multiply to this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.38.3.15 `__negative()`

```
GarbageCollected ComputedExpression::__negative ( ) const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.38.3.16 `__not()`

```
GarbageCollected ComputedExpression::__not ( ) const [virtual], [inherited]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.38.3.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<i>member</i>	The member expression provided by the script.
---------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.38.3.18 __slice()

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.38.3.19 __string()

```
GarbageCollected ComputedExpression::__string ( ) const [virtual], [inherited]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.38.3.20 __subtract()

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.38.3.21 dump()

```
string ComputedExpressionIterator::dump ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.38.3.22 is_equal() [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.38.3.23 `is_equal()` [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.38.3.24 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.38.3.25 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.38.3.26 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

val	The value to compare against.
-----	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.38.3.27 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

val	The value to compare against.
-----	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.38.3.28 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.38.3.29 `makeCopy()`

```
GarbageCollected ComputedExpression::makeCopy ( ) const [virtual], [inherited]
```

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionNativeLibraryFunction](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionLibraryMath](#), [Tang::ComputedExpressionLibrary](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), [Tang::ComputedExpressionBoolean](#), and [Tang::ComputedExpressionArray](#).

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

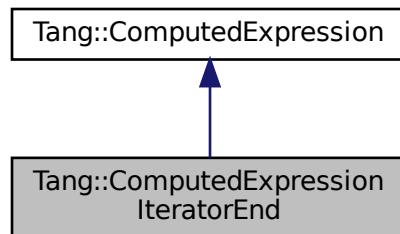
- [include/computedExpressionIterator.hpp](#)
- [src/computedExpressionIterator.cpp](#)

5.39 `Tang::ComputedExpressionIteratorEnd` Class Reference

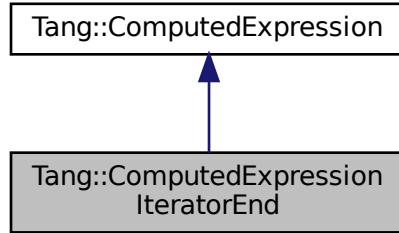
Represents that a collection has no more values through which to iterate.

```
#include <computedExpressionIteratorEnd.hpp>
```

Inheritance diagram for `Tang::ComputedExpressionIteratorEnd`:



Collaboration diagram for Tang::ComputedExpressionIteratorEnd:



Public Member Functions

- `ComputedExpressionIteratorEnd ()`
Construct an IteratorEnd result.
- `virtual std::string dump () const override`
Output the contents of the `ComputedExpression` as a string.
- `virtual GarbageCollected __string () const override`
Perform a type cast to string.
- `virtual std::string __asCode () const`
Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- `virtual bool isCopyNeeded () const`
Determine whether or not a copy is needed.
- `virtual GarbageCollected makeCopy () const`
Make a copy of the `ComputedExpression` (recursively, if appropriate).
- `virtual bool is_equal (const Tang::integer_t &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const Tang::float_t &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const bool &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const std::string &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const Error &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const std::nullptr_t &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual GarbageCollected __assign_index (const GarbageCollected &index, const GarbageCollected &value)`
Perform an index assignment to the supplied value.
- `virtual GarbageCollected __add (const GarbageCollected &rhs) const`
Compute the result of adding this value and the supplied value.
- `virtual GarbageCollected __subtract (const GarbageCollected &rhs) const`
Compute the result of subtracting this value and the supplied value.
- `virtual GarbageCollected __multiply (const GarbageCollected &rhs) const`
Compute the result of multiplying this value and the supplied value.

- virtual `GarbageCollected __divide (const GarbageCollected &rhs) const`
Compute the result of dividing this value and the supplied value.
- virtual `GarbageCollected __modulo (const GarbageCollected &rhs) const`
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __negative () const`
Compute the result of negating this value.
- virtual `GarbageCollected __not () const`
Compute the logical not of this value.
- virtual `GarbageCollected __lessThan (const GarbageCollected &rhs) const`
Compute the "less than" comparison.
- virtual `GarbageCollected __equal (const GarbageCollected &rhs) const`
Perform an equality test.
- virtual `GarbageCollected __period (const GarbageCollected &member, std::shared_ptr< TangBase > &tang) const`
Perform a member access (period) operation.
- virtual `GarbageCollected __index (const GarbageCollected &index) const`
Perform an index operation.
- virtual `GarbageCollected __slice (const GarbageCollected &begin, const GarbageCollected &end, const GarbageCollected &skip) const`
Perform a slice operation.
- virtual `GarbageCollected __getIterator (const GarbageCollected &collection) const`
Get an iterator for the expression.
- virtual `GarbageCollected __iteratorNext (size_t index=0) const`
Get the next iterative value.
- virtual `GarbageCollected __integer () const`
Perform a type cast to integer.
- virtual `GarbageCollected __float () const`
Perform a type cast to float.
- virtual `GarbageCollected __boolean () const`
Perform a type cast to boolean.

5.39.1 Detailed Description

Represents that a collection has no more values through which to iterate.

5.39.2 Member Function Documentation

5.39.2.1 `__add()`

```
GarbageCollected ComputedExpression::__add (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.39.2.2 [__asCode\(\)](#)

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.39.2.3 [__assign_index\(\)](#)

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.39.2.4 `__boolean()`

```
GarbageCollected ComputedExpression::__boolean ( ) const [virtual], [inherited]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.39.2.5 `__divide()`

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.39.2.6 `__equal()`

```
GarbageCollected ComputedExpression::__equal (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionNativeLibraryFunction](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), and [Tang::ComputedExpressionBoolean](#).

5.39.2.7 `__float()`

```
GarbageCollected ComputedExpression::__float () const [virtual], [inherited]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.39.2.8 `__getIterator()`

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<code>collection</code>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------------	--

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.39.2.9 `__index()`

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

<code>index</code>	The index expression provided by the script.
--------------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.39.2.10 `__integer()`

```
GarbageCollected ComputedExpression::__integer ( ) const [virtual], [inherited]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.39.2.11 `__iteratorNext()`

```
GarbageCollected ComputedExpression::__iteratorNext ( size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.39.2.12 `__lessThan()`

```
GarbageCollected ComputedExpression::__lessThan ( const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.39.2.13 `__modulo()`

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.39.2.14 `__multiply()`

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.39.2.15 `__negative()`

```
GarbageCollected ComputedExpression::__negative ( ) const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.39.2.16 `__not()`

```
GarbageCollected ComputedExpression::__not ( ) const [virtual], [inherited]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.39.2.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<code>member</code>	The member expression provided by the script.
---------------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.39.2.18 __slice()

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.39.2.19 __string()

```
GarbageCollected ComputedExpressionIteratorEnd::__string ( ) const [override], [virtual]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.39.2.20 __subtract()

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.39.2.21 dump()

```
string ComputedExpressionIteratorEnd::dump ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.39.2.22 is_equal() [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.39.2.23 `is_equal()` [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.39.2.24 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.39.2.25 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.39.2.26 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

val	The value to compare against.
-----	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.39.2.27 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

val	The value to compare against.
-----	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.39.2.28 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.39.2.29 `makeCopy()`

```
GarbageCollected ComputedExpression::makeCopy ( ) const [virtual], [inherited]
```

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionNativeLibraryFunction](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionLibraryMath](#), [Tang::ComputedExpressionLibrary](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), [Tang::ComputedExpressionBoolean](#), and [Tang::ComputedExpressionArray](#).

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

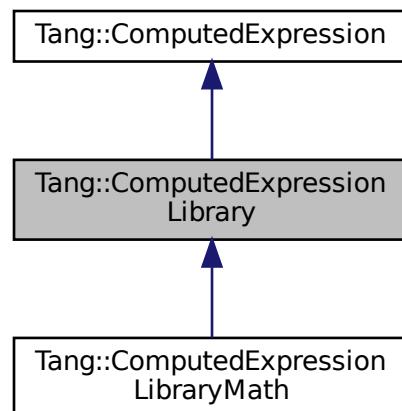
- [include/computedExpressionIteratorEnd.hpp](#)
- [src/computedExpressionIteratorEnd.cpp](#)

5.40 `Tang::ComputedExpressionLibrary` Class Reference

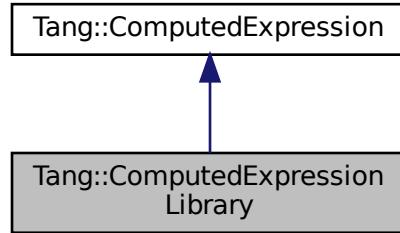
Represents a Runtime Library.

```
#include <computedExpressionLibrary.hpp>
```

Inheritance diagram for `Tang::ComputedExpressionLibrary`:



Collaboration diagram for Tang::ComputedExpressionLibrary:



Public Member Functions

- **ComputedExpressionLibrary ()**
Construct a Runtime Library.
- **virtual std::string dump () const override**
Output the contents of the [ComputedExpression](#) as a string.
- **GarbageCollected makeCopy () const override**
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- **GarbageCollected __period (const GarbageCollected &member, std::shared_ptr< TangBase > &tang) const override**
Perform a member access (period) operation.
- **virtual std::string __asCode () const**
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- **virtual bool isCopyNeeded () const**
Determine whether or not a copy is needed.
- **virtual bool is_equal (const Tang::integer_t &val) const**
Check whether or not the computed expression is equal to another value.
- **virtual bool is_equal (const Tang::float_t &val) const**
Check whether or not the computed expression is equal to another value.
- **virtual bool is_equal (const bool &val) const**
Check whether or not the computed expression is equal to another value.
- **virtual bool is_equal (const std::string &val) const**
Check whether or not the computed expression is equal to another value.
- **virtual bool is_equal (const Error &val) const**
Check whether or not the computed expression is equal to another value.
- **virtual bool is_equal (const std::nullptr_t &val) const**
Check whether or not the computed expression is equal to another value.
- **virtual GarbageCollected __assign_index (const GarbageCollected &index, const GarbageCollected &value) const**
Perform an index assignment to the supplied value.
- **virtual GarbageCollected __add (const GarbageCollected &rhs) const**
Compute the result of adding this value and the supplied value.
- **virtual GarbageCollected __subtract (const GarbageCollected &rhs) const**
Compute the result of subtracting this value and the supplied value.
- **virtual GarbageCollected __multiply (const GarbageCollected &rhs) const**

- virtual `GarbageCollected __divide` (const `GarbageCollected &rhs`) const

Compute the result of dividing this value and the supplied value.
- virtual `GarbageCollected __modulo` (const `GarbageCollected &rhs`) const

Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __negative` () const

Compute the result of negating this value.
- virtual `GarbageCollected __not` () const

Compute the logical not of this value.
- virtual `GarbageCollected __lessThan` (const `GarbageCollected &rhs`) const

Compute the "less than" comparison.
- virtual `GarbageCollected __equal` (const `GarbageCollected &rhs`) const

Perform an equality test.
- virtual `GarbageCollected __index` (const `GarbageCollected &index`) const

Perform an index operation.
- virtual `GarbageCollected __slice` (const `GarbageCollected &begin`, const `GarbageCollected &end`, const `GarbageCollected &skip`) const

Perform a slice operation.
- virtual `GarbageCollected __getIterator` (const `GarbageCollected &collection`) const

Get an iterator for the expression.
- virtual `GarbageCollected __iteratorNext` (size_t index=0) const

Get the next iterative value.
- virtual `GarbageCollected __integer` () const

Perform a type cast to integer.
- virtual `GarbageCollected __float` () const

Perform a type cast to float.
- virtual `GarbageCollected __boolean` () const

Perform a type cast to boolean.
- virtual `GarbageCollected __string` () const

Perform a type cast to string.

5.40.1 Detailed Description

Represents a Runtime Library.

5.40.2 Member Function Documentation

5.40.2.1 `__add()`

```
GarbageCollected ComputedExpression::__add (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.40.2.2 [__asCode\(\)](#)

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.40.2.3 [__assign_index\(\)](#)

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.40.2.4 `__boolean()`

```
GarbageCollected ComputedExpression::__boolean ( ) const [virtual], [inherited]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.40.2.5 `__divide()`

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.40.2.6 `__equal()`

```
GarbageCollected ComputedExpression::__equal (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionNativeLibraryFunction](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), and [Tang::ComputedExpressionBoolean](#).

5.40.2.7 `__float()`

`GarbageCollected` `ComputedExpression::__float () const [virtual], [inherited]`

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.40.2.8 `__getIterator()`

`GarbageCollected` `ComputedExpression::__getIterator (`
`const GarbageCollected & collection) const [virtual], [inherited]`

Get an iterator for the expression.

Parameters

<code>collection</code>	The <code>GarbageCollected</code> value that will serve as the collection through which to iterate.
-------------------------	---

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.40.2.9 `__index()`

`GarbageCollected` `ComputedExpression::__index (`
`const GarbageCollected & index) const [virtual], [inherited]`

Perform an index operation.

Parameters

<code>index</code>	The index expression provided by the script.
--------------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.40.2.10 __integer()

```
GarbageCollected ComputedExpression::__integer ( ) const [virtual], [inherited]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.40.2.11 __iteratorNext()

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.40.2.12 __lessThan()

```
GarbageCollected ComputedExpression::__lessThan (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.40.2.13 `__modulo()`

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.40.2.14 `__multiply()`

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.40.2.15 `__negative()`

```
GarbageCollected ComputedExpression::__negative ( ) const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.40.2.16 `__not()`

```
GarbageCollected ComputedExpression::__not ( ) const [virtual], [inherited]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.40.2.17 `__period()`

```
GarbageCollected ComputedExpressionLibrary::__period ( 
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [override], [virtual]
```

Perform a member access (period) operation.

Parameters

<code>member</code>	The member expression provided by the script.
---------------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.40.2.18 __slice()

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.40.2.19 __string()

```
GarbageCollected ComputedExpression::__string ( ) const [virtual], [inherited]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.40.2.20 __subtract()

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.40.2.21 dump()

```
std::string ComputedExpressionLibrary::dump ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.40.2.22 is_equal() [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.40.2.23 `is_equal()` [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.40.2.24 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.40.2.25 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.40.2.26 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

val	The value to compare against.
-----	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.40.2.27 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

val	The value to compare against.
-----	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.40.2.28 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.40.2.29 `makeCopy()`

`GarbageCollected` `ComputedExpressionLibrary::makeCopy () const [override], [virtual]`

Make a copy of the `ComputedExpression` (recursively, if appropriate).

Returns

A `Tang::GarbageCollected` value for the new `ComputedExpression`.

Reimplemented from `Tang::ComputedExpression`.

Reimplemented in `Tang::ComputedExpressionLibraryMath`.

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

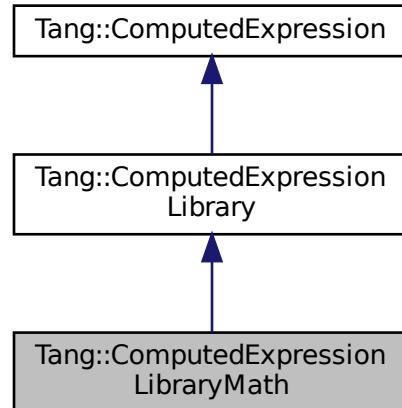
- `include/computedExpressionLibrary.hpp`
- `src/computedExpressionLibrary.cpp`

5.41 `Tang::ComputedExpressionLibraryMath` Class Reference

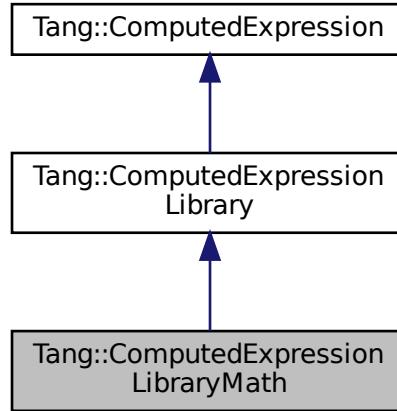
Represents a Runtime LibraryMath.

```
#include <computedExpressionLibraryMath.hpp>
```

Inheritance diagram for `Tang::ComputedExpressionLibraryMath`:



Collaboration diagram for Tang::ComputedExpressionLibraryMath:



Public Member Functions

- [ComputedExpressionLibraryMath \(\)](#)
Construct a Runtime Library for Math operations.
- [GarbageCollected makeCopy \(\) const override](#)
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- [virtual std::string dump \(\) const override](#)
Output the contents of the [ComputedExpression](#) as a string.
- [GarbageCollected __period \(const GarbageCollected &member, std::shared_ptr< TangBase > &tang\) const override](#)
Perform a member access (period) operation.
- [virtual std::string __asCode \(\) const](#)
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- [virtual bool isCopyNeeded \(\) const](#)
Determine whether or not a copy is needed.
- [virtual bool is_equal \(const Tang::integer_t &val\) const](#)
Check whether or not the computed expression is equal to another value.
- [virtual bool is_equal \(const Tang::float_t &val\) const](#)
Check whether or not the computed expression is equal to another value.
- [virtual bool is_equal \(const bool &val\) const](#)
Check whether or not the computed expression is equal to another value.
- [virtual bool is_equal \(const std::string &val\) const](#)
Check whether or not the computed expression is equal to another value.
- [virtual bool is_equal \(const Error &val\) const](#)
Check whether or not the computed expression is equal to another value.
- [virtual bool is_equal \(const std::nullptr_t &val\) const](#)
Check whether or not the computed expression is equal to another value.
- [virtual GarbageCollected __assign_index \(const GarbageCollected &index, const GarbageCollected &value\) const](#)
Perform an index assignment to the supplied value.

- virtual `GarbageCollected __add` (const `GarbageCollected` &rhs) const
Compute the result of adding this value and the supplied value.
- virtual `GarbageCollected __subtract` (const `GarbageCollected` &rhs) const
Compute the result of subtracting this value and the supplied value.
- virtual `GarbageCollected __multiply` (const `GarbageCollected` &rhs) const
Compute the result of multiplying this value and the supplied value.
- virtual `GarbageCollected __divide` (const `GarbageCollected` &rhs) const
Compute the result of dividing this value and the supplied value.
- virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __negative` () const
Compute the result of negating this value.
- virtual `GarbageCollected __not` () const
Compute the logical not of this value.
- virtual `GarbageCollected __lessThan` (const `GarbageCollected` &rhs) const
Compute the "less than" comparison.
- virtual `GarbageCollected __equal` (const `GarbageCollected` &rhs) const
Perform an equality test.
- virtual `GarbageCollected __index` (const `GarbageCollected` &index) const
Perform an index operation.
- virtual `GarbageCollected __slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const
Perform a slice operation.
- virtual `GarbageCollected __getIterator` (const `GarbageCollected` &collection) const
Get an iterator for the expression.
- virtual `GarbageCollected __iteratorNext` (size_t index=0) const
Get the next iterative value.
- virtual `GarbageCollected __integer` () const
Perform a type cast to integer.
- virtual `GarbageCollected __float` () const
Perform a type cast to float.
- virtual `GarbageCollected __boolean` () const
Perform a type cast to boolean.
- virtual `GarbageCollected __string` () const
Perform a type cast to string.

Static Public Member Functions

- static `LibraryFunctionMap getLibraryAttributes` ()
Construct the LibarayFunctionMap for this map's attributes.

5.41.1 Detailed Description

Represents a Runtime LibraryMath.

5.41.2 Member Function Documentation

5.41.2.1 `__add()`

```
GarbageCollected ComputedExpression::__add (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to add to this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.41.2.2 `__asCode()`

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.41.2.3 `__assign_index()`

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.41.2.4 `__boolean()`

```
GarbageCollected ComputedExpression::__boolean ( ) const [virtual], [inherited]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.41.2.5 `__divide()`

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of dividing this value and the supplied value.

Parameters

<code>rhs</code>	The GarbageCollected value to divide this by.
------------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.41.2.6 `__equal()`

```
GarbageCollected ComputedExpression::__equal (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Perform an equality test.

Parameters

<code>rhs</code>	The GarbageCollected value to compare against.
------------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionNativeLibraryFunction](#), [Tang::ComputedExpression](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), and [Tang::ComputedExpressionBoolean](#).

5.41.2.7 [__float\(\)](#)

```
GarbageCollected ComputedExpression::__float ( ) const [virtual], [inherited]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.41.2.8 [__getIterator\(\)](#)

```
GarbageCollected ComputedExpression::__getIterator ( const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<code>collection</code>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------------	--

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.41.2.9 [__index\(\)](#)

```
GarbageCollected ComputedExpression::__index ( const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.41.2.10 __integer()

`GarbageCollected ComputedExpression::__integer () const [virtual], [inherited]`

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.41.2.11 __iteratorNext()

`GarbageCollected ComputedExpression::__iteratorNext (size_t index = 0) const [virtual], [inherited]`

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.41.2.12 __lessThan()

`GarbageCollected ComputedExpression::__lessThan (const GarbageCollected & rhs) const [virtual], [inherited]`

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.41.2.13 [__modulo\(\)](#)

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.41.2.14 [__multiply\(\)](#)

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.41.2.15 `__negative()`

```
GarbageCollected ComputedExpression::__negative ( ) const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.41.2.16 `__not()`

```
GarbageCollected ComputedExpression::__not ( ) const [virtual], [inherited]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.41.2.17 `__period()`

```
GarbageCollected ComputedExpressionLibrary::__period ( 
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [override], [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<code>member</code>	The member expression provided by the script.
---------------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.41.2.18 __slice()

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.41.2.19 __string()

```
GarbageCollected ComputedExpression::__string () const [virtual], [inherited]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.41.2.20 __subtract()

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.41.2.21 `dump()`

```
std::string ComputedExpressionLibrary::dump ( ) const [override], [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.41.2.22 `getLibraryAttributes()`

```
LibraryFunctionMap ComputedExpressionLibraryMath::getLibraryAttributes ( ) [static]
```

Construct the LibararyFunctionMap for this map's attributes.

Returns

The library attribute functions for this library.

5.41.2.23 `is_equal()` [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.41.2.24 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.41.2.25 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.41.2.26 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.41.2.27 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.41.2.28 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.41.2.29 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.41.2.30 makeCopy()

```
GarbageCollected ComputedExpressionLibraryMath::makeCopy ( ) const [override], [virtual]
```

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented from [Tang::ComputedExpressionLibrary](#).

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

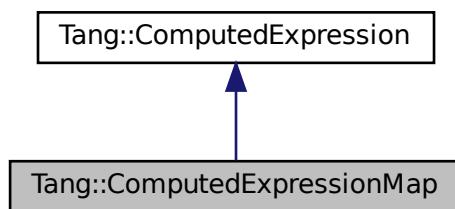
- [include/computedExpressionLibraryMath.hpp](#)
- [src/computedExpressionLibraryMath.cpp](#)

5.42 Tang::ComputedExpressionMap Class Reference

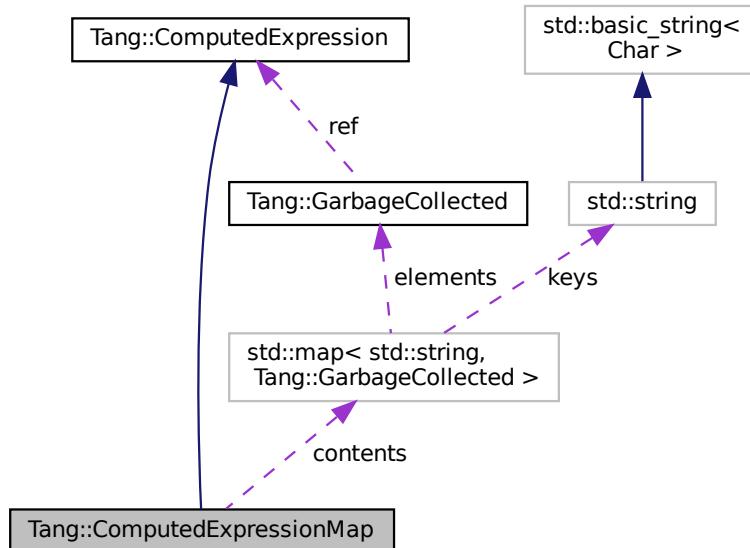
Represents an Map that is the result of a computation.

```
#include <computedExpressionMap.hpp>
```

Inheritance diagram for Tang::ComputedExpressionMap:



Collaboration diagram for Tang::ComputedExpressionMap:



Public Member Functions

- **ComputedExpressionMap (std::map< std::string, Tang::GarbageCollected > contents)**
Construct an Map result.
- virtual std::string **dump () const override**
Output the contents of the `ComputedExpression` as a string.
- virtual bool **isCopyNeeded () const override**
Determine whether or not a copy is needed.
- **GarbageCollected makeCopy () const override**
Make a copy of the `ComputedExpression` (recursively, if appropriate).
- virtual **GarbageCollected __index (const GarbageCollected &index) const override**
Perform an index operation.
- virtual **GarbageCollected __getIterator (const GarbageCollected &collection) const override**
Get an iterator for the expression.
- virtual **GarbageCollected __iteratorNext (size_t index) const override**
Get the next iterative value.
- virtual **GarbageCollected __assign_index (const GarbageCollected &index, const GarbageCollected &value) const override**
Perform an index assignment to the supplied value.
- virtual **GarbageCollected __string () const override**
Perform a type cast to string.
- virtual **GarbageCollected __boolean () const override**
Perform a type cast to boolean.
- virtual std::string **__asCode () const**
Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- virtual bool **is_equal (const Tang::integer_t &val) const**

- virtual bool `is_equal` (const `Tang::float_t` &val) const

Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const bool &val) const

Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const std::string &val) const

Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const `Error` &val) const

Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const std::nullptr_t &val) const

Check whether or not the computed expression is equal to another value.
- virtual `GarbageCollected __add` (const `GarbageCollected` &rhs) const

Compute the result of adding this value and the supplied value.
- virtual `GarbageCollected __subtract` (const `GarbageCollected` &rhs) const

Compute the result of subtracting this value and the supplied value.
- virtual `GarbageCollected __multiply` (const `GarbageCollected` &rhs) const

Compute the result of multiplying this value and the supplied value.
- virtual `GarbageCollected __divide` (const `GarbageCollected` &rhs) const

Compute the result of dividing this value and the supplied value.
- virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const

Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __negative` () const

Compute the result of negating this value.
- virtual `GarbageCollected __not` () const

Compute the logical not of this value.
- virtual `GarbageCollected __lessThan` (const `GarbageCollected` &rhs) const

Compute the "less than" comparison.
- virtual `GarbageCollected __equal` (const `GarbageCollected` &rhs) const

Perform an equality test.
- virtual `GarbageCollected __period` (const `GarbageCollected` &member, std::shared_ptr< `TangBase` > &tang) const

Perform a member access (period) operation.
- virtual `GarbageCollected __slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const

Perform a slice operation.
- virtual `GarbageCollected __integer` () const

Perform a type cast to integer.
- virtual `GarbageCollected __float` () const

Perform a type cast to float.

Private Attributes

- std::map< std::string, `Tang::GarbageCollected` > `contents`

The map contents.

5.42.1 Detailed Description

Represents an Map that is the result of a computation.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 ComputedExpressionMap()

```
ComputedExpressionMap::ComputedExpressionMap (   
    std::map< std::string, Tang::GarbageCollected > contents )
```

Construct an Map result.

Parameters

<i>contents</i>	The map of key value pairs.
-----------------	-----------------------------

5.42.3 Member Function Documentation

5.42.3.1 __add()

```
GarbageCollected ComputedExpression::__add (   
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to add to this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.42.3.2 __asCode()

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the **ComputedExpression** as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.42.3.3 __assign_index()

```
GarbageCollected ComputedExpressionMap::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [override], [virtual]
```

Perform an index assignment to the supplied value.

Parameters

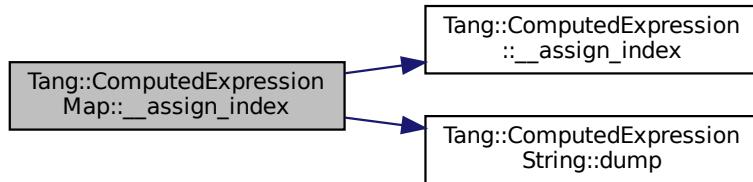
<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.42.3.4 __boolean()

```
GarbageCollected ComputedExpressionMap::__boolean ( ) const [override], [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.42.3.5 __divide()

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.42.3.6 [__equal\(\)](#)

```
GarbageCollected ComputedExpression::__equal (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionNativeLibraryFunction](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), and [Tang::ComputedExpressionBoolean](#).

5.42.3.7 [__float\(\)](#)

```
GarbageCollected ComputedExpression::__float () const [virtual], [inherited]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.42.3.8 [__getIterator\(\)](#)

```
GarbageCollected ComputedExpressionMap::__getIterator (
    const GarbageCollected & collection ) const [override], [virtual]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

Reimplemented from [Tang::ComputedExpression](#).

5.42.3.9 `__index()`

```
GarbageCollected ComputedExpressionMap::__index (
    const GarbageCollected & index ) const [override], [virtual]
```

Perform an index operation.

Parameters

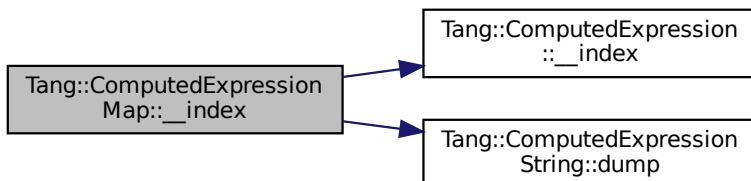
<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.42.3.10 `__integer()`

```
GarbageCollected ComputedExpression::__integer ( ) const [virtual], [inherited]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.42.3.11 `__iteratorNext()`

```
GarbageCollected ComputedExpressionMap::__iteratorNext (
    size_t index ) const [override], [virtual]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented from [Tang::ComputedExpression](#).

5.42.3.12 `__lessThan()`

```
GarbageCollected ComputedExpression::__lessThan (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.42.3.13 `__modulo()`

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.42.3.14 `__multiply()`

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<code>rhs</code>	The GarbageCollected value to multiply to this.
------------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.42.3.15 `__negative()`

```
GarbageCollected ComputedExpression::__negative () const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.42.3.16 `__not()`

```
GarbageCollected ComputedExpression::__not () const [virtual], [inherited]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.42.3.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<i>member</i>	The member expression provided by the script.
---------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.42.3.18 `__slice()`

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.42.3.19 `__string()`

```
GarbageCollected ComputedExpressionMap::__string ( ) const [override], [virtual]
```

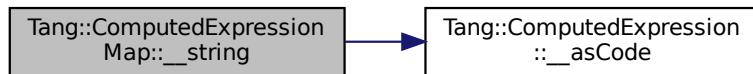
Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.42.3.20 `__subtract()`

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.42.3.21 `dump()`

```
string ComputedExpressionMap::dump ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.42.3.22 `is_equal()` [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.42.3.23 `is_equal()` [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.42.3.24 `is_equal()` [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

5.42.3.25 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.42.3.26 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.42.3.27 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.42.3.28 isCopyNeeded()

```
bool ComputedExpressionMap::isCopyNeeded ( ) const [override], [virtual]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented from [Tang::ComputedExpression](#).

5.42.3.29 makeCopy()

```
GarbageCollected ComputedExpressionMap::makeCopy ( ) const [override], [virtual]
```

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented from [Tang::ComputedExpression](#).

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

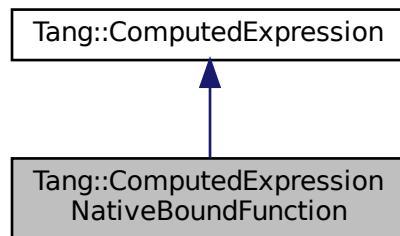
- [include/computedExpressionMap.hpp](#)
- [src/computedExpressionMap.cpp](#)

5.43 Tang::ComputedExpressionNativeBoundFunction Class Reference

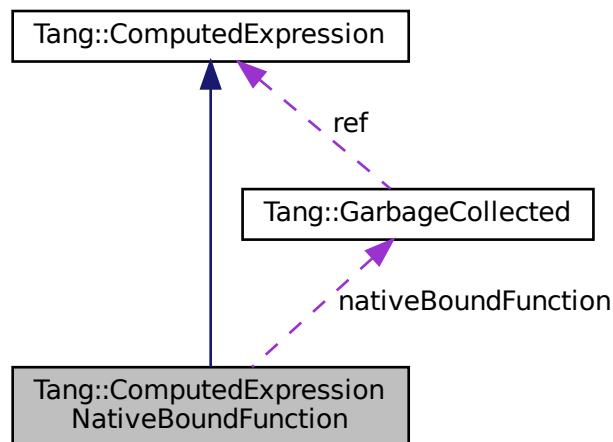
Represents a NativeBound Function declared in the script.

```
#include <computedExpressionNativeBoundFunction.hpp>
```

Inheritance diagram for Tang::ComputedExpressionNativeBoundFunction:



Collaboration diagram for Tang::ComputedExpressionNativeBoundFunction:



Public Member Functions

- [ComputedExpressionNativeBoundFunction](#) (NativeBoundFunction nativeBoundFunction, size_t argc, std::type_index targetTypeIndex)
Construct an NativeBoundFunction.
- virtual std::string [dump](#) () const override

- **Output the contents of the `ComputedExpression` as a string.**
- **GarbageCollected `makeCopy` () const override**
 - Make a copy of the `ComputedExpression` (recursively, if appropriate).*
- **virtual `GarbageCollected __equal` (const `GarbageCollected` &rhs) const override**
 - Perform an equality test.*
- **NativeBoundFunction `getFunction` () const**
 - Get the native bound function to be executed.*
- **size_t `getArgc` () const**
 - Get the count of arguments that this function expects.*
- **const std::type_index & `getTargetTypeIndex` () const**
 - Get the type of the value to which the function is bound.*
- **virtual std::string `__asCode` () const**
 - Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.*
- **virtual bool `isCopyNeeded` () const**
 - Determine whether or not a copy is needed.*
- **virtual bool `is_equal` (const `Tang::integer_t` &val) const**
 - Check whether or not the computed expression is equal to another value.*
- **virtual bool `is_equal` (const `Tang::float_t` &val) const**
 - Check whether or not the computed expression is equal to another value.*
- **virtual bool `is_equal` (const bool &val) const**
 - Check whether or not the computed expression is equal to another value.*
- **virtual bool `is_equal` (const std::string &val) const**
 - Check whether or not the computed expression is equal to another value.*
- **virtual bool `is_equal` (const `Error` &val) const**
 - Check whether or not the computed expression is equal to another value.*
- **virtual bool `is_equal` (const std::nullptr_t &val) const**
 - Check whether or not the computed expression is equal to another value.*
- **virtual `GarbageCollected __assign_index` (const `GarbageCollected` &index, const `GarbageCollected` &value) const**
 - Perform an index assignment to the supplied value.*
- **virtual `GarbageCollected __add` (const `GarbageCollected` &rhs) const**
 - Compute the result of adding this value and the supplied value.*
- **virtual `GarbageCollected __subtract` (const `GarbageCollected` &rhs) const**
 - Compute the result of subtracting this value and the supplied value.*
- **virtual `GarbageCollected __multiply` (const `GarbageCollected` &rhs) const**
 - Compute the result of multiplying this value and the supplied value.*
- **virtual `GarbageCollected __divide` (const `GarbageCollected` &rhs) const**
 - Compute the result of dividing this value and the supplied value.*
- **virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const**
 - Compute the result of moduloing this value and the supplied value.*
- **virtual `GarbageCollected __negative` () const**
 - Compute the result of negating this value.*
- **virtual `GarbageCollected __not` () const**
 - Compute the logical not of this value.*
- **virtual `GarbageCollected __lessThan` (const `GarbageCollected` &rhs) const**
 - Compute the "less than" comparison.*
- **virtual `GarbageCollected __period` (const `GarbageCollected` &member, std::shared_ptr<`TangBase`> &tang) const**
 - Perform a member access (period) operation.*
- **virtual `GarbageCollected __index` (const `GarbageCollected` &index) const**
 - Perform an index operation.*

- virtual `GarbageCollected __slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const
Perform a slice operation.
- virtual `GarbageCollected __getIterator` (const `GarbageCollected` &collection) const
Get an iterator for the expression.
- virtual `GarbageCollected __iteratorNext` (size_t index=0) const
Get the next iterative value.
- virtual `GarbageCollected __integer` () const
Perform a type cast to integer.
- virtual `GarbageCollected __float` () const
Perform a type cast to float.
- virtual `GarbageCollected __boolean` () const
Perform a type cast to boolean.
- virtual `GarbageCollected __string` () const
Perform a type cast to string.

Public Attributes

- `std::optional<GarbageCollected> target`
The target object that the function is bound to.

Private Attributes

- `NativeBoundFunction nativeBoundFunction`
The native bound function to be executed.
- `size_t argc`
The count of arguments that this function expects.
- `std::type_index targetTypeIndex`
The type of the value to which the function is bound.

5.43.1 Detailed Description

Represents a NativeBound Function declared in the script.

5.43.2 Constructor & Destructor Documentation

5.43.2.1 ComputedExpressionNativeBoundFunction()

```
ComputedExpressionNativeBoundFunction::ComputedExpressionNativeBoundFunction (
    NativeBoundFunction nativeBoundFunction,
    size_t argc,
    std::type_index targetTypeIndex )
```

Construct an NativeBoundFunction.

The object itself is designed to be safe in that, once it is constructed, the method function pointer, argument count, and target type cannot be changed, but can only be accessible through a getter.

The target value that the function is bound to, however, cannot be set when the object is created, due to the design of the compiler. It is therefore exposed, regardless of being made public or via a setter function.

The current design of the VM will set the correct target, but because the target is exposed, it is possible that some bad actor could modify it. It is therefore necessary to verify that the type of the bound object and the type that was known when this object is created are, in fact, the same. That is why we store the target object type information and protect it behind a getter function.

When the VM executes the bound method, it will perform a type check to verify that the bound object is of the same type as that of the method that is defined in [TangBase::getObjectMethods\(\)](#).

It should be safe, then, to assume that within a NativeBoundFunction, the type is the expected type. No [ComputedExpression](#) type, then, should "steal" a NativeBoundFunction from another [ComputedExpression](#) definition, as it is assumed that the bound target that is provided to any NativeBoundFunction is the same as the type on which it was originally defined.

For example, a NativeBoundFunction declared in [ComputedExpressionString](#) may assume that the bound target is also a [ComputedExpressionString](#). If another class, such as [ComputedExpressionArray](#), were to try to copy the NativeBoundFunction (as a pointer reference), the function will still expect that the bound target is a [ComputedExpressionString](#), and will probably cause a segmentation fault. Just don't do it.

Parameters

<i>nativeBoundFunction</i>	The native bound function to be executed.
<i>argc</i>	The count of arguments that this function expects.
<i>targetTypeIndex</i>	The type of the value to which the function is bound.

5.43.3 Member Function Documentation

5.43.3.1 __add()

```
GarbageCollected ComputedExpression::__add (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.43.3.2 [__asCode\(\)](#)

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.43.3.3 [__assign_index\(\)](#)

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.43.3.4 `__boolean()`

```
GarbageCollected ComputedExpression::__boolean ( ) const [virtual], [inherited]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.43.3.5 `__divide()`

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.43.3.6 `__equal()`

```
GarbageCollected ComputedExpressionNativeBoundFunction::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

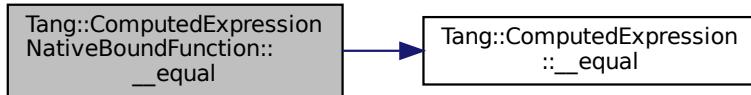
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.43.3.7 __float()

[GarbageCollected](#) `ComputedExpression::__float () const [virtual], [inherited]`

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.43.3.8 __getIterator()

[GarbageCollected](#) `ComputedExpression::__getIterator (`
`const GarbageCollected & collection) const [virtual], [inherited]`

Get an iterator for the expression.

Parameters

<code>collection</code>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------------	--

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.43.3.9 __index()

[GarbageCollected](#) `ComputedExpression::__index (`
`const GarbageCollected & index) const [virtual], [inherited]`

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.43.3.10 `__integer()`

```
GarbageCollected ComputedExpression::__integer ( ) const [virtual], [inherited]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.43.3.11 `__iteratorNext()`

```
GarbageCollected ComputedExpression::__iteratorNext ( size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.43.3.12 `__lessThan()`

```
GarbageCollected ComputedExpression::__lessThan ( const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.43.3.13 [__modulo\(\)](#)

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.43.3.14 [__multiply\(\)](#)

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.43.3.15 `__negative()`

```
GarbageCollected ComputedExpression::__negative ( ) const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.43.3.16 `__not()`

```
GarbageCollected ComputedExpression::__not ( ) const [virtual], [inherited]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.43.3.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<code>member</code>	The member expression provided by the script.
---------------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.43.3.18 `__slice()`

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<code>begin</code>	The begin index expression provided by the script.
<code>end</code>	The end index expression provided by the script.
<code>skip</code>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.43.3.19 `__string()`

```
GarbageCollected ComputedExpression::__string ( ) const [virtual], [inherited]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.43.3.20 `__subtract()`

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.43.3.21 dump()

```
string ComputedExpressionNativeBoundFunction::dump ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.43.3.22 getArgc()

```
size_t ComputedExpressionNativeBoundFunction::getArgc ( ) const
```

Get the count of arguments that this function expects.

Returns

The count of arguments that this function expects.

5.43.3.23 getFunction()

```
NativeBoundFunction ComputedExpressionNativeBoundFunction::getFunction ( ) const
```

Get the native bound function to be executed.

Returns

The native bound function to be executed.

5.43.3.24 `getTargetTypeIndex()`

```
const type_index & ComputedExpressionNativeBoundFunction::getTargetTypeIndex ( ) const
```

Get the type of the value to which the function is bound.

Returns

The type of the value to which the function is bound.

5.43.3.25 `is_equal()` [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.43.3.26 `is_equal()` [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.43.3.27 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.43.3.28 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.43.3.29 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.43.3.30 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.43.3.31 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.43.3.32 makeCopy()

```
GarbageCollected ComputedExpressionNativeBoundFunction::makeCopy ( ) const [override], [virtual]
```

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented from [Tang::ComputedExpression](#).

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

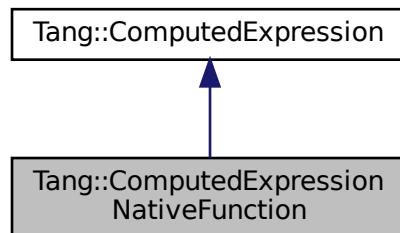
- [include/computedExpressionNativeBoundFunction.hpp](#)
- [src/computedExpressionNativeBoundFunction.cpp](#)

5.44 Tang::ComputedExpressionNativeFunction Class Reference

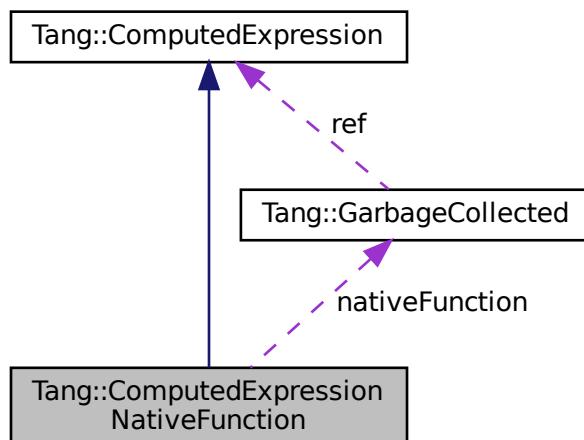
Represents a Native Function provided by compiled C++ code.

```
#include <computedExpressionNativeFunction.hpp>
```

Inheritance diagram for Tang::ComputedExpressionNativeFunction:



Collaboration diagram for Tang::ComputedExpressionNativeFunction:



Public Member Functions

- [ComputedExpressionNativeFunction \(NativeFunction nativeFunction, size_t argc\)](#)
Construct an NativeFunction.
- virtual std::string [dump \(\) const override](#)
Output the contents of the [ComputedExpression](#) as a string.

- `GarbageCollected makeCopy () const override`
Make a copy of the `ComputedExpression` (recursively, if appropriate).
- `virtual GarbageCollected __equal (const GarbageCollected &rhs) const override`
Perform an equality test.
- `NativeFunction getFunction () const`
Get the native bound function to be executed.
- `size_t getArgc () const`
Get the count of arguments that this function expects.
- `virtual std::string __asCode () const`
Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- `virtual bool isCopyNeeded () const`
Determine whether or not a copy is needed.
- `virtual bool is_equal (const Tang::integer_t &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const Tang::float_t &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const bool &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const std::string &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const Error &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual bool is_equal (const std::nullptr_t &val) const`
Check whether or not the computed expression is equal to another value.
- `virtual GarbageCollected __assign_index (const GarbageCollected &index, const GarbageCollected &value) const`
Perform an index assignment to the supplied value.
- `virtual GarbageCollected __add (const GarbageCollected &rhs) const`
Compute the result of adding this value and the supplied value.
- `virtual GarbageCollected __subtract (const GarbageCollected &rhs) const`
Compute the result of subtracting this value and the supplied value.
- `virtual GarbageCollected __multiply (const GarbageCollected &rhs) const`
Compute the result of multiplying this value and the supplied value.
- `virtual GarbageCollected __divide (const GarbageCollected &rhs) const`
Compute the result of dividing this value and the supplied value.
- `virtual GarbageCollected __modulo (const GarbageCollected &rhs) const`
Compute the result of moduloing this value and the supplied value.
- `virtual GarbageCollected __negative () const`
Compute the result of negating this value.
- `virtual GarbageCollected __not () const`
Compute the logical not of this value.
- `virtual GarbageCollected __lessThan (const GarbageCollected &rhs) const`
Compute the "less than" comparison.
- `virtual GarbageCollected __period (const GarbageCollected &member, std::shared_ptr< TangBase > &tang) const`
Perform a member access (period) operation.
- `virtual GarbageCollected __index (const GarbageCollected &index) const`
Perform an index operation.
- `virtual GarbageCollected __slice (const GarbageCollected &begin, const GarbageCollected &end, const GarbageCollected &skip) const`
Perform a slice operation.
- `virtual GarbageCollected __getIterator (const GarbageCollected &collection) const`

- `virtual GarbageCollected __iteratorNext (size_t index=0) const`
Get an iterator for the expression.
- `virtual GarbageCollected __integer () const`
Get the next iterative value.
- `virtual GarbageCollected __float () const`
Perform a type cast to float.
- `virtual GarbageCollected __boolean () const`
Perform a type cast to boolean.
- `virtual GarbageCollected __string () const`
Perform a type cast to string.

Private Attributes

- `NativeFunction nativeFunction`
The native bound function to be executed.
- `size_t argc`
The count of arguments that this function expects.

5.44.1 Detailed Description

Represents a Native Function provided by compiled C++ code.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 ComputedExpressionNativeFunction()

```
ComputedExpressionNativeFunction::ComputedExpressionNativeFunction (
    NativeFunction nativeFunction,
    size_t argc )
```

Construct an NativeFunction.

The object itself is designed to be safe in that, once it is constructed, the method function pointer, and argument count cannot be changed, but can only be accessible through a getter.

Parameters

<code>nativeFunction</code>	The native function to be executed.
<code>argc</code>	The count of arguments that this function expects.

5.44.3 Member Function Documentation

5.44.3.1 `__add()`

```
GarbageCollected ComputedExpression::__add (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to add to this.
------------	---

Returns

The result of the operation.

Reimplemented in `Tang::ComputedExpressionString`, `Tang::ComputedExpressionInteger`, `Tang::ComputedExpressionFloat`, and `Tang::ComputedExpressionError`.

5.44.3.2 `__asCode()`

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in `Tang::ComputedExpressionString`.

5.44.3.3 `__assign_index()`

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.44.3.4 `__boolean()`

```
GarbageCollected ComputedExpression::__boolean ( ) const [virtual], [inherited]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.44.3.5 `__divide()`

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.44.3.6 `__equal()`

```
GarbageCollected ComputedExpressionNativeFunction::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:

**5.44.3.7 __float()**

[GarbageCollected](#) `ComputedExpression::__float () const [virtual], [inherited]`

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.44.3.8 __getIterator()

[GarbageCollected](#) `ComputedExpression::__getIterator (`
`const GarbageCollected & collection) const [virtual], [inherited]`

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.44.3.9 `__index()`

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.44.3.10 `__integer()`

```
GarbageCollected ComputedExpression::__integer () const [virtual], [inherited]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.44.3.11 `__iteratorNext()`

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.44.3.12 `__lessThan()`

```
GarbageCollected ComputedExpression::__lessThan (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the "less than" comparison.

Parameters

<code>rhs</code>	The GarbageCollected value to compare against.
------------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.44.3.13 `__modulo()`

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<code>rhs</code>	The GarbageCollected value to modulo this by.
------------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.44.3.14 `__multiply()`

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<code>rhs</code>	The GarbageCollected value to multiply to this.
------------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.44.3.15 `__negative()`

[GarbageCollected](#) `ComputedExpression::__negative () const [virtual], [inherited]`

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.44.3.16 `__not()`

[GarbageCollected](#) `ComputedExpression::__not () const [virtual], [inherited]`

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.44.3.17 `__period()`

[GarbageCollected](#) `ComputedExpression::__period (`
`const GarbageCollected & member,`
`std::shared_ptr< TangBase > & tang) const [virtual], [inherited]`

Perform a member access (period) operation.

Parameters

<i>member</i>	The member expression provided by the script.
---------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.44.3.18 `__slice()`

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.44.3.19 `__string()`

```
GarbageCollected ComputedExpression::__string ( ) const [virtual], [inherited]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.44.3.20 `__subtract()`

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The <code>GarbageCollected</code> value to subtract from this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.44.3.21 `dump()`

```
string ComputedExpressionNativeFunction::dump ( ) const [override], [virtual]
```

Output the contents of the `ComputedExpression` as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.44.3.22 `getArgc()`

```
size_t ComputedExpressionNativeFunction::getArgc ( ) const
```

Get the count of arguments that this function expects.

Returns

The count of arguments that this function expects.

5.44.3.23 `getFunction()`

```
NativeFunction ComputedExpressionNativeFunction::getFunction ( ) const
```

Get the native bound function to be executed.

Returns

The native bound function to be executed.

5.44.3.24 `is_equal()` [1/6]

```
virtual bool Tang::ComputedExpression::is_equal ( const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.44.3.25 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.44.3.26 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.44.3.27 `is_equal()` [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.44.3.28 `is_equal()` [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.44.3.29 `is_equal()` [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.44.3.30 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.44.3.31 makeCopy()

```
GarbageCollected ComputedExpressionNativeFunction::makeCopy ( ) const [override], [virtual]
```

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented from [Tang::ComputedExpression](#).

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

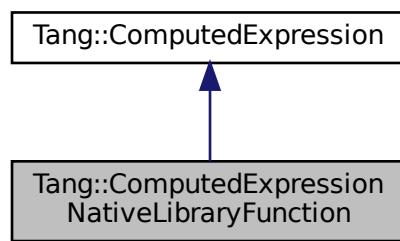
- [include/computedExpressionNativeFunction.hpp](#)
- [src/computedExpressionNativeFunction.cpp](#)

5.45 Tang::ComputedExpressionNativeLibraryFunction Class Reference

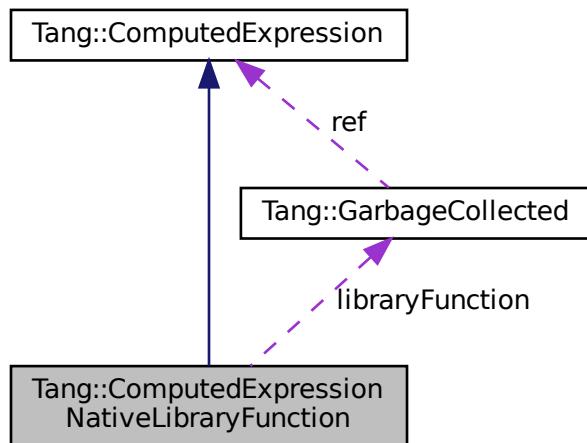
Represents a Native Function provided by compiled C++ code that is executed to create a library or one of its attributes.

```
#include <computedExpressionNativeLibraryFunction.hpp>
```

Inheritance diagram for Tang::ComputedExpressionNativeLibraryFunction:



Collaboration diagram for Tang::ComputedExpressionNativeLibraryFunction:



Public Member Functions

- [ComputedExpressionNativeLibraryFunction \(LibraryFunction nativeFunction\)](#)
Construct a NativeLibraryFunction.
- [virtual std::string dump \(\) const override](#)

- **Output the contents of the `ComputedExpression` as a string.**
 - **GarbageCollected `makeCopy` () const override**
 - Make a copy of the `ComputedExpression` (recursively, if appropriate).*
 - **virtual GarbageCollected `__equal` (const GarbageCollected &rhs) const override**
 - Perform an equality test.*
 - **LibraryFunction `getFunction` () const**
 - Get the native bound function to be executed.*
 - **virtual std::string `__asCode` () const**
 - Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.*
 - **virtual bool `isCopyNeeded` () const**
 - Determine whether or not a copy is needed.*
 - **virtual bool `is_equal` (const Tang::integer_t &val) const**
 - Check whether or not the computed expression is equal to another value.*
 - **virtual bool `is_equal` (const Tang::float_t &val) const**
 - Check whether or not the computed expression is equal to another value.*
 - **virtual bool `is_equal` (const bool &val) const**
 - Check whether or not the computed expression is equal to another value.*
 - **virtual bool `is_equal` (const std::string &val) const**
 - Check whether or not the computed expression is equal to another value.*
 - **virtual bool `is_equal` (const Error &val) const**
 - Check whether or not the computed expression is equal to another value.*
 - **virtual bool `is_equal` (const std::nullptr_t &val) const**
 - Check whether or not the computed expression is equal to another value.*
 - **virtual GarbageCollected `__assign_index` (const GarbageCollected &index, const GarbageCollected &value)**
 - Perform an index assignment to the supplied value.*
 - **virtual GarbageCollected `__add` (const GarbageCollected &rhs) const**
 - Compute the result of adding this value and the supplied value.*
 - **virtual GarbageCollected `__subtract` (const GarbageCollected &rhs) const**
 - Compute the result of subtracting this value and the supplied value.*
 - **virtual GarbageCollected `__multiply` (const GarbageCollected &rhs) const**
 - Compute the result of multiplying this value and the supplied value.*
 - **virtual GarbageCollected `__divide` (const GarbageCollected &rhs) const**
 - Compute the result of dividing this value and the supplied value.*
 - **virtual GarbageCollected `__modulo` (const GarbageCollected &rhs) const**
 - Compute the result of moduloing this value and the supplied value.*
 - **virtual GarbageCollected `__negative` () const**
 - Compute the result of negating this value.*
 - **virtual GarbageCollected `__not` () const**
 - Compute the logical not of this value.*
 - **virtual GarbageCollected `__lessThan` (const GarbageCollected &rhs) const**
 - Compute the "less than" comparison.*
 - **virtual GarbageCollected `__period` (const GarbageCollected &member, std::shared_ptr< TangBase > &tang) const**
 - Perform a member access (period) operation.*
 - **virtual GarbageCollected `__index` (const GarbageCollected &index) const**
 - Perform an index operation.*
 - **virtual GarbageCollected `__slice` (const GarbageCollected &begin, const GarbageCollected &end, const GarbageCollected &skip) const**
 - Perform a slice operation.*
 - **virtual GarbageCollected `__getIterator` (const GarbageCollected &collection) const**
 - Get an iterator for the expression.*

- virtual `GarbageCollected __iteratorNext (size_t index=0) const`
Get the next iterative value.
- virtual `GarbageCollected __integer () const`
Perform a type cast to integer.
- virtual `GarbageCollected __float () const`
Perform a type cast to float.
- virtual `GarbageCollected __boolean () const`
Perform a type cast to boolean.
- virtual `GarbageCollected __string () const`
Perform a type cast to string.

Private Attributes

- `LibraryFunction libraryFunction`
The library function to be executed.

5.45.1 Detailed Description

Represents a Native Function provided by compiled C++ code that is executed to create a library or one of its attributes.

The purpose of this function is to be able to construct a library or the library attributes as needed at runtime.

5.45.2 Constructor & Destructor Documentation

5.45.2.1 `ComputedExpressionNativeLibraryFunction()`

```
ComputedExpressionNativeLibraryFunction::ComputedExpressionNativeLibraryFunction (
    LibraryFunction nativeFunction )
```

Construct a NativeLibraryFunction.

Parameters

<code>libraryFunction</code>	The library function to be executed.
------------------------------	--------------------------------------

5.45.3 Member Function Documentation

5.45.3.1 `__add()`

```
GarbageCollected ComputedExpression::__add (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.45.3.2 [__asCode\(\)](#)

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.45.3.3 [__assign_index\(\)](#)

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.45.3.4 `__boolean()`

```
GarbageCollected ComputedExpression::__boolean ( ) const [virtual], [inherited]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.45.3.5 `__divide()`

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.45.3.6 `__equal()`

```
GarbageCollected ComputedExpressionNativeLibraryFunction::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

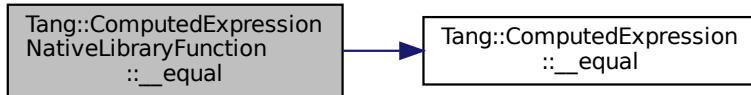
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.45.3.7 __float()

[GarbageCollected](#) `ComputedExpression::__float () const [virtual], [inherited]`

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.45.3.8 __getIterator()

[GarbageCollected](#) `ComputedExpression::__getIterator (const GarbageCollected & collection) const [virtual], [inherited]`

Get an iterator for the expression.

Parameters

<code>collection</code>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------------	--

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.45.3.9 __index()

[GarbageCollected](#) `ComputedExpression::__index (const GarbageCollected & index) const [virtual], [inherited]`

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.45.3.10 `__integer()`

```
GarbageCollected ComputedExpression::__integer ( ) const [virtual], [inherited]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.45.3.11 `__iteratorNext()`

```
GarbageCollected ComputedExpression::__iteratorNext ( size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.45.3.12 `__lessThan()`

```
GarbageCollected ComputedExpression::__lessThan ( const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.45.3.13 [__modulo\(\)](#)

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.45.3.14 [__multiply\(\)](#)

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.45.3.15 `__negative()`

```
GarbageCollected ComputedExpression::__negative ( ) const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.45.3.16 `__not()`

```
GarbageCollected ComputedExpression::__not ( ) const [virtual], [inherited]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.45.3.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<code>member</code>	The member expression provided by the script.
---------------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.45.3.18 `__slice()`

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<code>begin</code>	The begin index expression provided by the script.
<code>end</code>	The end index expression provided by the script.
<code>skip</code>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.45.3.19 `__string()`

```
GarbageCollected ComputedExpression::__string ( ) const [virtual], [inherited]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), [Tang::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.45.3.20 `__subtract()`

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.45.3.21 `dump()`

```
string ComputedExpressionNativeLibraryFunction::dump ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.45.3.22 `getFunction()`

```
LibraryFunction ComputedExpressionNativeLibraryFunction::getFunction ( ) const
```

Get the native bound function to be executed.

Returns

The native bound function to be executed.

5.45.3.23 `is_equal()` [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.45.3.24 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.45.3.25 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

5.45.3.26 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.45.3.27 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.45.3.28 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.45.3.29 `isCopyNeeded()`

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.45.3.30 `makeCopy()`

```
GarbageCollected ComputedExpressionNativeLibraryFunction::makeCopy ( ) const [override], [virtual]
```

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented from [Tang::ComputedExpression](#).

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

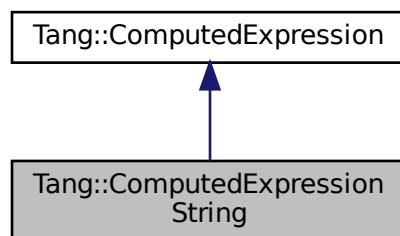
- [include/computedExpressionNativeLibraryFunction.hpp](#)
- [src/computedExpressionNativeLibraryFunction.cpp](#)

5.46 Tang::ComputedExpressionString Class Reference

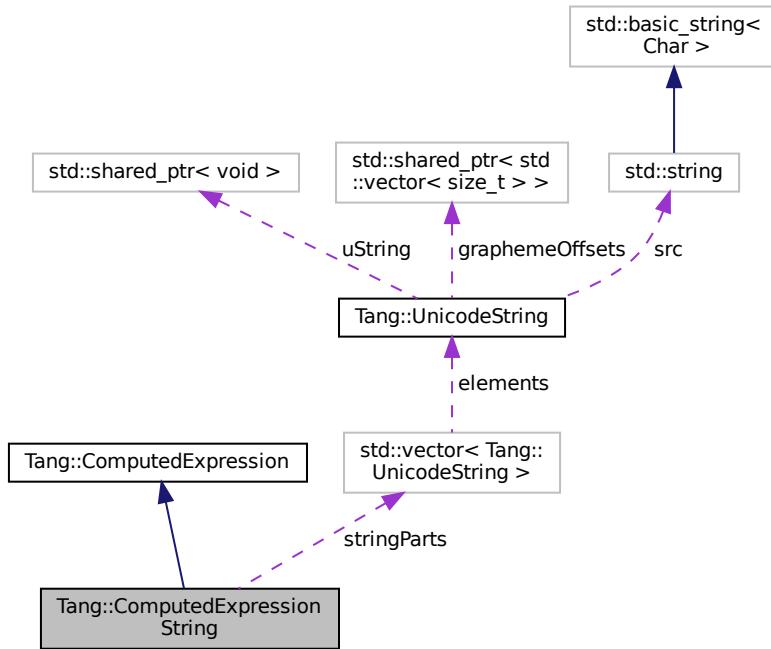
Represents a String that is the result of a computation.

```
#include <computedExpressionString.hpp>
```

Inheritance diagram for Tang::ComputedExpressionString:



Collaboration diagram for Tang::ComputedExpressionString:



Public Member Functions

- `ComputedExpressionString (const std::string &val)`
Construct a String result.
- `ComputedExpressionString (const std::vector<UnicodeString> &stringParts)`
Construct a String result from a vector of `UnicodeString` objects.
- virtual `std::string dump () const override`
Output the contents of the `ComputedExpression` as a string.
- virtual `std::string __asCode () const override`
Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- `GarbageCollected makeCopy () const override`
Make a copy of the `ComputedExpression` (recursively, if appropriate).
- virtual `bool is_equal (const bool &val) const override`
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal (const std::string &val) const override`
Check whether or not the computed expression is equal to another value.
- virtual `GarbageCollected __index (const GarbageCollected &index) const override`
Perform an index operation.
- virtual `GarbageCollected __slice (const GarbageCollected &begin, const GarbageCollected &end, const GarbageCollected &skip) const override`
Perform a slice operation.
- virtual `GarbageCollected __getIterator (const GarbageCollected &collection) const override`
Get an iterator for the expression.
- virtual `GarbageCollected __iteratorNext (size_t index) const override`

- Get the next iterative value.*
- virtual `GarbageCollected __add` (const `GarbageCollected` &rhs) const override
Compute the result of adding this value and the supplied value.
 - virtual `GarbageCollected __not` () const override
Compute the logical not of this value.
 - virtual `GarbageCollected __lessThan` (const `GarbageCollected` &rhs) const override
Compute the "less than" comparison.
 - virtual `GarbageCollected __equal` (const `GarbageCollected` &rhs) const override
Perform an equality test.
 - virtual `GarbageCollected __boolean` () const override
Perform a type cast to boolean.
 - virtual `GarbageCollected __string` () const override
Perform a type cast to string.
 - const std::vector< `UnicodeString` > & `getValue` () const
Return the collection of string values that are stored in this object.
 - size_t `length` () const
Return the number of graphemes contained in the string.
 - size_t `bytesLength` () const
Return the number of bytes required by the string, stored as UTF-8.
 - `ComputedExpressionString` & `operator+=` (const `ComputedExpressionString` &rhs)
Helper function to copy the contents of the rhs string into the current string.
 - void `setUntrusted` ()
Set all of the string parts to untrusted.
 - virtual bool `isCopyNeeded` () const
Determine whether or not a copy is needed.
 - virtual bool `is_equal` (const `Tang::integer_t` &val) const
Check whether or not the computed expression is equal to another value.
 - virtual bool `is_equal` (const `Tang::float_t` &val) const
Check whether or not the computed expression is equal to another value.
 - virtual bool `is_equal` (const `Error` &val) const
Check whether or not the computed expression is equal to another value.
 - virtual bool `is_equal` (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
 - virtual `GarbageCollected __assign_index` (const `GarbageCollected` &index, const `GarbageCollected` &value)
Perform an index assignment to the supplied value.
 - virtual `GarbageCollected __subtract` (const `GarbageCollected` &rhs) const
Compute the result of subtracting this value and the supplied value.
 - virtual `GarbageCollected __multiply` (const `GarbageCollected` &rhs) const
Compute the result of multiplying this value and the supplied value.
 - virtual `GarbageCollected __divide` (const `GarbageCollected` &rhs) const
Compute the result of dividing this value and the supplied value.
 - virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const
Compute the result of moduloing this value and the supplied value.
 - virtual `GarbageCollected __negative` () const
Compute the result of negating this value.
 - virtual `GarbageCollected __period` (const `GarbageCollected` &member, std::shared_ptr< `TangBase` > &tang) const
Perform a member access (period) operation.
 - virtual `GarbageCollected __integer` () const
Perform a type cast to integer.
 - virtual `GarbageCollected __float` () const
Perform a type cast to float.

Static Public Member Functions

- static [NativeBoundFunctionMap getMethods \(\)](#)
Return the member functions implemented for this particular expression type.

Private Attributes

- std::vector< [UnicodeString](#) > [stringParts](#)
The string value.
- std::optional< size_t > [cachedLength](#)
Cache of the string length in graphemes.
- std::optional< size_t > [cachedBytesLength](#)
Cache of the string length in bytes.

5.46.1 Detailed Description

Represents a String that is the result of a computation.

5.46.2 Constructor & Destructor Documentation

5.46.2.1 [ComputedExpressionString\(\)](#) [1/2]

```
ComputedExpressionString::ComputedExpressionString (const std::string & val )
```

Construct a String result.

Parameters

<i>val</i>	The string value.
------------	-------------------

5.46.2.2 [ComputedExpressionString\(\)](#) [2/2]

```
ComputedExpressionString::ComputedExpressionString (const std::vector< UnicodeString > & stringParts )
```

Construct a String result from a vector of [UnicodeString](#) objects.

Parameters

<i>stringParts</i>	The vector of UnicodeString objects.
--------------------	--

5.46.3 Member Function Documentation

5.46.3.1 __add()

```
GarbageCollected ComputedExpressionString::__add (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of adding this value and the supplied value.

Parameters

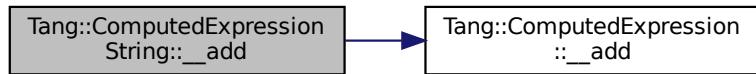
<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.46.3.2 __asCode()

```
string ComputedExpressionString::__asCode ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.46.3.3 __assign_index()

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.46.3.4 __boolean()

```
GarbageCollected ComputedExpressionString::__boolean () const [override], [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:

**5.46.3.5 __divide()**

```
GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.46.3.6 __equal()

```
GarbageCollected ComputedExpressionString::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

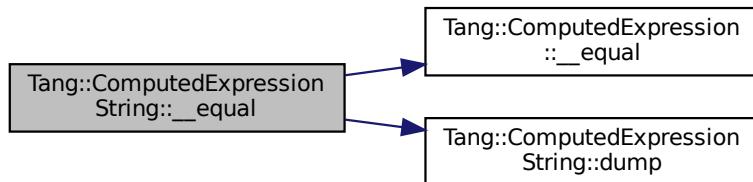
<i>rhs</i>	The GarbageCollected value to compare against.
------------	---

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:

**5.46.3.7 __float()**

[GarbageCollected](#) `ComputedExpression::__float () const [virtual], [inherited]`

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.46.3.8 __getIterator()

[GarbageCollected](#) `ComputedExpressionString::__getIterator (const GarbageCollected & collection) const [override], [virtual]`

Get an iterator for the expression.

Parameters

<code>collection</code>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------------	--

Reimplemented from [Tang::ComputedExpression](#).

5.46.3.9 __index()

```
GarbageCollected ComputedExpressionString::__index (
    const GarbageCollected & index ) const [override], [virtual]
```

Perform an index operation.

Parameters

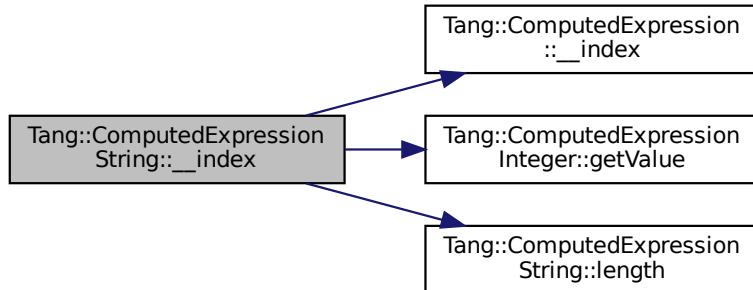
<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.46.3.10 __integer()

```
GarbageCollected ComputedExpression::__integer ( ) const [virtual], [inherited]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.46.3.11 `__iteratorNext()`

```
GarbageCollected ComputedExpressionString::__iteratorNext (
    size_t index ) const [override], [virtual]
```

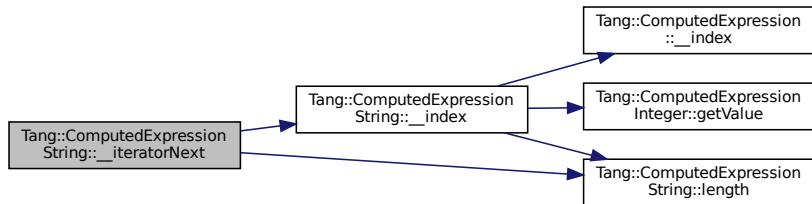
Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.46.3.12 `__lessThan()`

```
GarbageCollected ComputedExpressionString::__lessThan (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the "less than" comparison.

Parameters

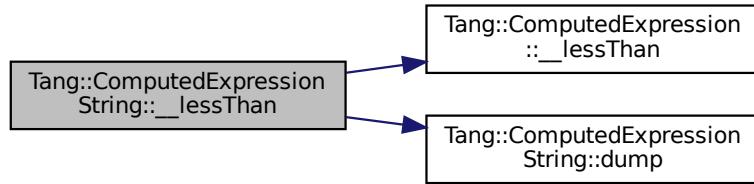
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.46.3.13 __modulo()

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.46.3.14 __multiply()

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.46.3.15 `__negative()`

```
GarbageCollected ComputedExpression::__negative ( ) const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.46.3.16 `__not()`

```
GarbageCollected ComputedExpressionString::__not ( ) const [override], [virtual]
```

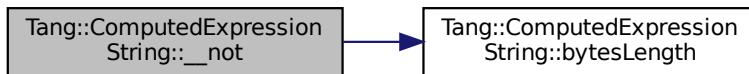
Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.46.3.17 `__period()`

```
GarbageCollected ComputedExpression::__period (
    const GarbageCollected & member,
    std::shared_ptr< TangBase > & tang ) const [virtual], [inherited]
```

Perform a member access (period) operation.

Parameters

<i>member</i>	The member expression provided by the script.
---------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionLibrary](#).

5.46.3.18 `__slice()`

```
GarbageCollected ComputedExpressionString::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [override], [virtual]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

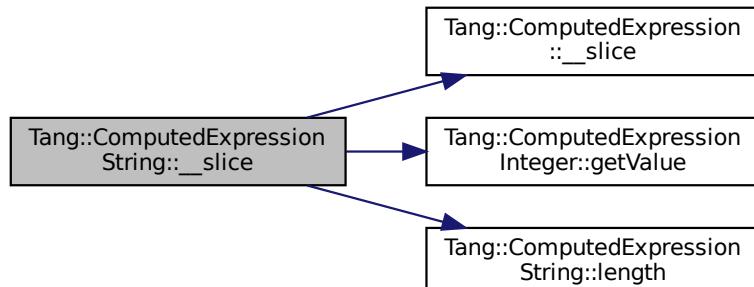
<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.46.3.19 `__string()`

```
GarbageCollected ComputedExpressionString::__string ( ) const [override], [virtual]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.46.3.20 `__subtract()`

```
GarbageCollected ComputedExpression::__subtract ( const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.46.3.21 `bytesLength()`

```
size_t ComputedExpressionString::bytesLength ( ) const
```

Return the number of bytes required by the string, stored as UTF-8.

Returns

The number of bytes required by the string, stored as UTF-8.

5.46.3.22 `dump()`

```
string ComputedExpressionString::dump ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.46.3.23 `getMethods()`

```
NativeBoundFunctionMap ComputedExpressionString::getMethods ( ) [static]
```

Return the member functions implemented for this particular expression type.

Returns

The member functions implemented.

Here is the call graph for this function:



5.46.3.24 `getValue()`

```
const vector< UnicodeString > & ComputedExpressionString::getValue ( ) const
```

Return the collection of string values that are stored in this object.

Returns

The collection of string values.

5.46.3.25 `is_equal()` [1/6]

```
bool ComputedExpressionString::is_equal (
    const bool & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

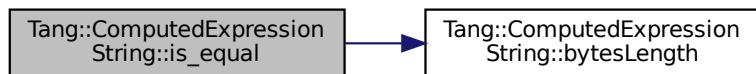
<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.46.3.26 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal ( const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.46.3.27 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal ( const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.46.3.28 `is_equal()` [4/6]

```
bool ComputedExpressionString::is_equal (
    const std::string & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:

**5.46.3.29 `is_equal()` [5/6]**

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.46.3.30 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.46.3.31 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.46.3.32 `length()`

```
size_t ComputedExpressionString::length ( ) const
```

Return the number of graphemes contained in the string.

Returns

The number of graphemes contained in the string.

5.46.3.33 `makeCopy()`

```
GarbageCollected ComputedExpressionString::makeCopy ( ) const [override], [virtual]
```

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented from [Tang::ComputedExpression](#).

5.46.3.34 `operator+=()`

```
ComputedExpressionString & ComputedExpressionString::operator+= ( const ComputedExpressionString & rhs )
```

Helper function to copy the contents of the rhs string into the current string.

Parameters

<i>rhs</i>	The right hand side of the operation.
------------	---------------------------------------

Returns

The result of the operation.

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

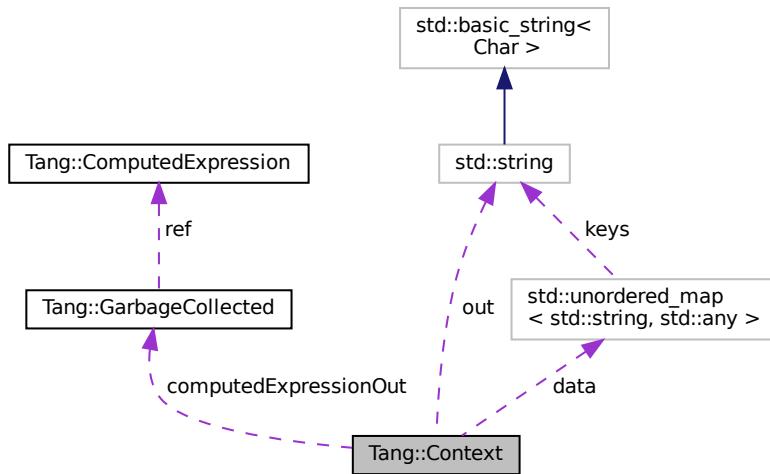
- [include/computedExpressionString.hpp](#)
- [src/computedExpressionString.cpp](#)

5.47 Tang::Context Class Reference

Holds all environment variables specific to the execution of a program.

```
#include <context.hpp>
```

Collaboration diagram for Tang::Context:



Public Member Functions

- [Context \(ContextData &&data\)](#)

Default constructor.

Public Attributes

- [ContextData data](#)
Holds arbitrary data for use in the program execution.
- [std::string out](#)
The output result from the program execution.
- [GarbageCollected computedExpressionOut](#)
The output result from the program execution, as a `ComputedExpressionString`.
- [std::optional< GarbageCollected > result](#)
The result of the `Program` execution.

5.47.1 Detailed Description

Holds all environment variables specific to the execution of a program.

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

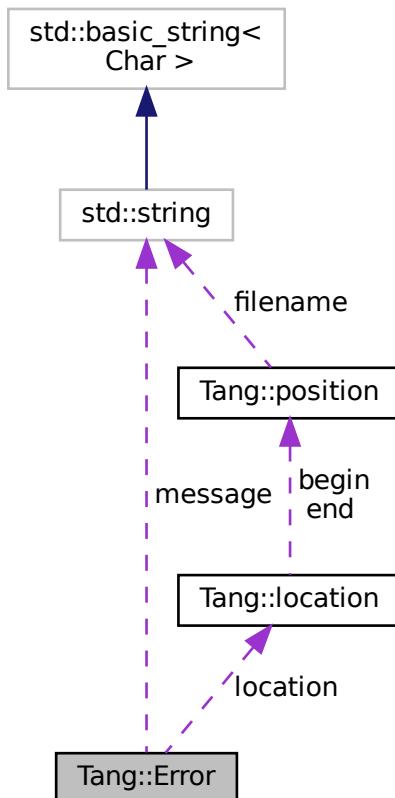
- `include/context.hpp`
- `src/context.cpp`

5.48 Tang::Error Class Reference

The [Error](#) class is used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error.

```
#include <error.hpp>
```

Collaboration diagram for Tang::Error:



Public Member Functions

- [Error \(\)](#)
Creates an empty error message.
- [Error \(std::string message\)](#)
Creates an error message using the supplied error string and location.
- [Error \(std::string message, Tang::location location\)](#)
Creates an error message using the supplied error string and location.

Public Attributes

- std::string [message](#)
The error message as a string.
- Tang::location [location](#)
The location of the error.

Friends

- std::ostream & `operator<<` (std::ostream &out, const `Error` &error)
Add friendly output.

5.48.1 Detailed Description

The `Error` class is used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error.

5.48.2 Constructor & Destructor Documentation

5.48.2.1 `Error()` [1/2]

```
Tang::Error::Error (
    std::string message )  [inline]
```

Creates an error message using the supplied error string and location.

Parameters

<code>message</code>	The error message as a string.
----------------------	--------------------------------

5.48.2.2 `Error()` [2/2]

```
Tang::Error::Error (
    std::string message,
    Tang::location location )  [inline]
```

Creates an error message using the supplied error string and location.

Parameters

<code>message</code>	The error message as a string.
<code>location</code>	The location of the error.

5.48.3 Friends And Related Function Documentation

5.48.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & out,
    const Error & error ) [friend]
```

Add friendly output.

Parameters

<i>out</i>	The output stream.
<i>error</i>	The Error object.

Returns

The output stream.

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

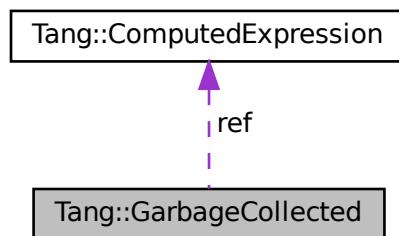
- [include/error.hpp](#)
- [src/error.cpp](#)

5.49 Tang::GarbageCollected Class Reference

A container that acts as a resource-counting garbage collector for the specified type.

```
#include <garbageCollected.hpp>
```

Collaboration diagram for Tang::GarbageCollected:



Public Member Functions

- `GarbageCollected (const GarbageCollected &other)`
Copy Constructor.
- `GarbageCollected (GarbageCollected &&other)`
Move Constructor.
- `GarbageCollected & operator= (const GarbageCollected &other)`
Copy Assignment.
- `GarbageCollected & operator= (GarbageCollected &&other)`
Move Assignment.
- `~GarbageCollected ()`
Destructor.
- `bool isCopyNeeded () const`
Determine whether or not a copy is needed as determined by the referenced `ComputedExpression`.
- `GarbageCollected makeCopy () const`
Create a separate copy of the original `GarbageCollected` value.
- `ComputedExpression * operator-> () const`
Access the tracked object as a pointer.
- `ComputedExpression & operator* () const`
Access the tracked object.
- `bool operator== (const Tang::integer_t &val) const`
Compare the `GarbageCollected` tracked object with a supplied value.
- `bool operator== (const Tang::float_t &val) const`
Compare the `GarbageCollected` tracked object with a supplied value.
- `bool operator== (const bool &val) const`
Compare the `GarbageCollected` tracked object with a supplied value.
- `bool operator== (const std::string &val) const`
Compare the `GarbageCollected` tracked object with a supplied value.
- `bool operator== (const char *const &val) const`
Compare the `GarbageCollected` tracked object with a supplied value.
- `bool operator== (const Error &val) const`
Compare the `GarbageCollected` tracked object with a supplied value.
- `bool operator== (const std::nullptr_t &null) const`
Compare the `GarbageCollected` tracked object with a supplied value.
- `GarbageCollected operator+ (const GarbageCollected &rhs) const`
Perform an addition between two `GarbageCollected` values.
- `GarbageCollected operator- (const GarbageCollected &rhs) const`
Perform a subtraction between two `GarbageCollected` values.
- `GarbageCollected operator* (const GarbageCollected &rhs) const`
Perform a multiplication between two `GarbageCollected` values.
- `GarbageCollected operator/ (const GarbageCollected &rhs) const`
Perform a division between two `GarbageCollected` values.
- `GarbageCollected operator% (const GarbageCollected &rhs) const`
Perform a modulo between two `GarbageCollected` values.
- `GarbageCollected operator- () const`
Perform a negation on the `GarbageCollected` value.
- `GarbageCollected operator! () const`
Perform a logical not on the `GarbageCollected` value.
- `GarbageCollected operator< (const GarbageCollected &rhs) const`
Perform a < between two `GarbageCollected` values.
- `GarbageCollected operator<= (const GarbageCollected &rhs) const`

- `GarbageCollected operator>` (const `GarbageCollected` &rhs) const
`Perform a > between two GarbageCollected values.`
- `GarbageCollected operator>=` (const `GarbageCollected` &rhs) const
`Perform a >= between two GarbageCollected values.`
- `GarbageCollected operator==` (const `GarbageCollected` &rhs) const
`Perform a == between two GarbageCollected values.`
- `GarbageCollected operator!=` (const `GarbageCollected` &rhs) const
`Perform a != between two GarbageCollected values.`

Static Public Member Functions

- `template<class T , typename... Args>`
`static GarbageCollected make (Args... args)`
`Creates a garbage-collected object of the specified type.`

Protected Member Functions

- `GarbageCollected ()`
`Constructs a garbage-collected object of the specified type.`

Protected Attributes

- `size_t * count`
`The count of references to the tracked object.`
- `ComputedExpression * ref`
`A reference to the tracked object.`
- `std::function< void(void)> recycle`
`A cleanup function to recycle the object.`

Friends

- `std::ostream & operator<<` (std::ostream &out, const `GarbageCollected` &gc)
`Add friendly output.`

5.49.1 Detailed Description

A container that acts as a resource-counting garbage collector for the specified type.

Uses the `SingletonObjectPool` to created and recycle object memory. The container is not thread-safe.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 `GarbageCollected()` [1/3]

```
GarbageCollected::GarbageCollected (
    const GarbageCollected & other )
```

Copy Constructor.

Parameters

<i>The</i>	other GarbageCollected object to copy.
------------	--

5.49.2.2 GarbageCollected() [2/3]

```
GarbageCollected::GarbageCollected (
    GarbageCollected && other )
```

Move Constructor.

Parameters

<i>The</i>	other GarbageCollected object to move.
------------	--

5.49.2.3 ~GarbageCollected()

```
GarbageCollected::~GarbageCollected ( )
```

Destructor.

Clean up the tracked object, if appropriate.

5.49.2.4 GarbageCollected() [3/3]

```
Tang::GarbageCollected::GarbageCollected ( ) [inline], [protected]
```

Constructs a garbage-collected object of the specified type.

It is private so that a [GarbageCollected](#) object can only be created using the [GarbageCollected::make\(\)](#) function.

Parameters

<i>variable</i>	The arguments to pass to the constructor of the specified type.
-----------------	---

5.49.3 Member Function Documentation**5.49.3.1 isCopyNeeded()**

```
bool GarbageCollected::isCopyNeeded ( ) const
```

Determine whether or not a copy is needed as determined by the referenced [ComputedExpression](#).

Returns

Whether or not a copy is needed.

5.49.3.2 `make()`

```
template<class T , typename... Args>
static GarbageCollected Tang::GarbageCollected::make (
    Args... args ) [inline], [static]
```

Creates a garbage-collected object of the specified type.

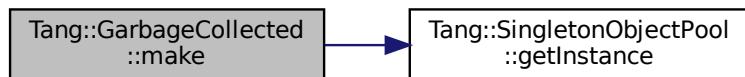
Parameters

<i>variable</i>	The arguments to pass to the constructor of the specified type.
-----------------	---

Returns

A [GarbageCollected](#) object.

Here is the call graph for this function:



5.49.3.3 `makeCopy()`

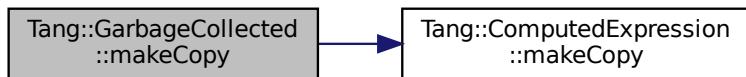
```
GarbageCollected GarbageCollected::makeCopy () const
```

Create a separate copy of the original [GarbageCollected](#) value.

Returns

A [GarbageCollected](#) copy of the original value.

Here is the call graph for this function:



5.49.3.4 operator"!"()

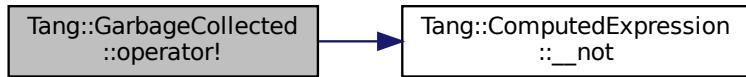
```
GarbageCollected GarbageCollected::operator! ( ) const
```

Perform a logical not on the [GarbageCollected](#) value.

Returns

The result of the operation.

Here is the call graph for this function:



5.49.3.5 operator"!=()

```
GarbageCollected GarbageCollected::operator!= ( const GarbageCollected & rhs ) const
```

Perform a != between two [GarbageCollected](#) values.

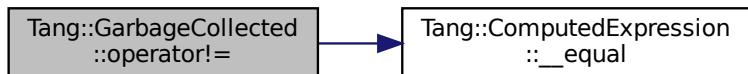
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:

**5.49.3.6 operator%()**

```
GarbageCollected GarbageCollected::operator% (
    const GarbageCollected & rhs ) const
```

Perform a modulo between two `GarbageCollected` values.

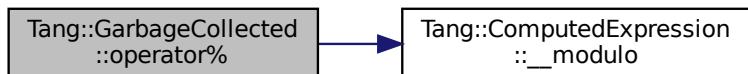
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.49.3.7 operator*() [1/2]

```
ComputedExpression & GarbageCollected::operator* ( ) const
```

Access the tracked object.

Returns

A reference to the tracked object.

5.49.3.8 operator*() [2/2]

```
GarbageCollected GarbageCollected::operator* (   
    const GarbageCollected & rhs ) const
```

Perform a multiplication between two **GarbageCollected** values.

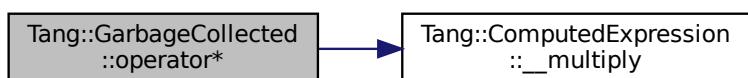
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.49.3.9 operator+()

```
GarbageCollected GarbageCollected::operator+ (   
    const GarbageCollected & rhs ) const
```

Perform an addition between two **GarbageCollected** values.

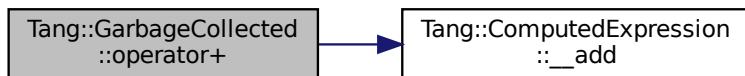
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.49.3.10 operator-() [1/2]

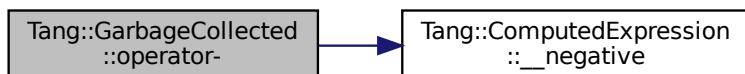
`GarbageCollected GarbageCollected::operator- () const`

Perform a negation on the `GarbageCollected` value.

Returns

The result of the operation.

Here is the call graph for this function:



5.49.3.11 operator-() [2/2]

`GarbageCollected GarbageCollected::operator- (const GarbageCollected & rhs) const`

Perform a subtraction between two `GarbageCollected` values.

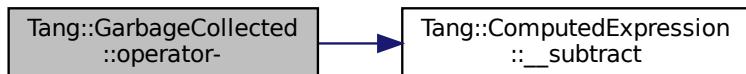
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.49.3.12 operator->()

```
ComputedExpression * GarbageCollected::operator-> ( ) const
```

Access the tracked object as a pointer.

Returns

A pointer to the tracked object.

5.49.3.13 operator/()

```
GarbageCollected GarbageCollected::operator/ ( const GarbageCollected & rhs ) const
```

Perform a division between two [GarbageCollected](#) values.

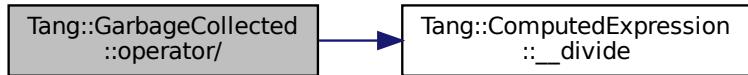
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.49.3.14 operator<()

```
GarbageCollected GarbageCollected::operator< (
    const GarbageCollected & rhs ) const
```

Perform a $<$ between two [GarbageCollected](#) values.

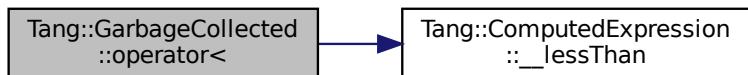
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.49.3.15 operator<=()

```
GarbageCollected GarbageCollected::operator<= (
    const GarbageCollected & rhs ) const
```

Perform a \leq between two [GarbageCollected](#) values.

Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

5.49.3.16 operator=() [1/2]

```
GarbageCollected & GarbageCollected::operator= (
    const GarbageCollected & other )
```

Copy Assignment.

Parameters

<i>The</i>	other GarbageCollected object.
------------	--------------------------------

5.49.3.17 operator=() [2/2]

```
GarbageCollected & GarbageCollected::operator= (
    GarbageCollected && other )
```

Move Assignment.

Parameters

<i>The</i>	other GarbageCollected object.
------------	--------------------------------

5.49.3.18 operator==() [1/8]

```
bool GarbageCollected::operator== (
    const bool & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

5.49.3.19 operator==() [2/8]

```
bool GarbageCollected::operator== (
    const char *const & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

5.49.3.20 operator==() [3/8]

```
bool GarbageCollected::operator== (
    const Error & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

5.49.3.21 operator==() [4/8]

```
GarbageCollected GarbageCollected::operator== (
    const GarbageCollected & rhs ) const
```

Perform a == between two [GarbageCollected](#) values.

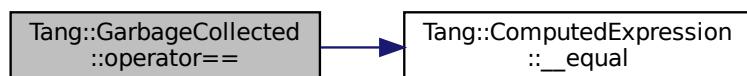
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:

**5.49.3.22 operator==() [5/8]**

```
bool GarbageCollected::operator== ( const std::nullptr_t & null ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

5.49.3.23 operator==() [6/8]

```
bool GarbageCollected::operator== ( const std::string & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

5.49.3.24 operator==(7/8)

```
bool GarbageCollected::operator== (
    const Tang::float_t & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

5.49.3.25 operator==(8/8)

```
bool GarbageCollected::operator== (
    const Tang::integer_t & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

Here is the call graph for this function:



5.49.3.26 operator>()

```
GarbageCollected GarbageCollected::operator> (
    const GarbageCollected & rhs ) const
```

Perform a `>` between two `GarbageCollected` values.

Parameters

<code>rhs</code>	The right hand side operand.
------------------	------------------------------

Returns

The result of the operation.

5.49.3.27 operator>=()

```
GarbageCollected GarbageCollected::operator>= (
    const GarbageCollected & rhs ) const
```

Perform a `>=` between two `GarbageCollected` values.

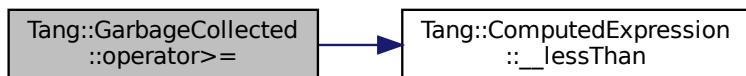
Parameters

<code>rhs</code>	The right hand side operand.
------------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.49.4 Friends And Related Function Documentation

5.49.4.1 `operator<<`

```
std::ostream& operator<< (  
    std::ostream & out,  
    const GarbageCollected & gc ) [friend]
```

Add friendly output.

Parameters

<code>out</code>	The output stream.
<code>gc</code>	The GarbageCollected value.

Returns

The output stream.

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

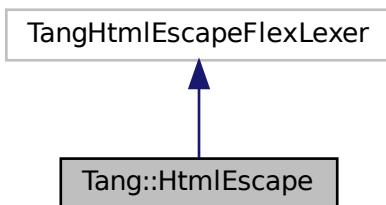
- [include/garbageCollected.hpp](#)
- [src/garbageCollected.cpp](#)

5.50 `Tang::HtmlEscape` Class Reference

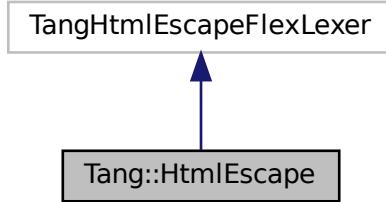
The Flex lexer class for the main Tang language.

```
#include <htmlEscape.hpp>
```

Inheritance diagram for `Tang::HtmlEscape`:



Collaboration diagram for Tang::HtmlEscape:



Public Member Functions

- [HtmlEscape](#) (std::istream &arg_yyin, std::ostream &arg_yyout)
The constructor for the Scanner.
- virtual std::string [get_next_token](#) ()
Extract the next token from the input string.

5.50.1 Detailed Description

The Flex lexer class for the main Tang language.

Flex requires that our lexer class inherit from yyFlexLexer, an "intermediate" class whose real name is "TangTangFlexLexer". We are subclassing it so that we can override the return type of [get_next_token\(\)](#), for compatibility with Bison 3 tokens.

5.50.2 Constructor & Destructor Documentation

5.50.2.1 HtmlEscape()

```
Tang::HtmlEscape::HtmlEscape (
    std::istream & arg_yyin,
    std::ostream & arg_yyout ) [inline]
```

The constructor for the Scanner.

The design of the Flex lexer is to tokenize the contents of an input stream, and to write any error messages to an output stream. In our implementation, however, errors are returned differently, so the output stream is never used. Its presence is retained, however, in case it is needed in the future.

For now, the general approach should be to supply the input as a string stream, and to use std::cout as the output.

Parameters

<i>arg_yyin</i>	The input stream to be tokenized
<i>arg_yyout</i>	The output stream (not currently used)

5.50.3 Member Function Documentation**5.50.3.1 `get_next_token()`**

```
virtual std::string Tang::HtmlEscape::get_next_token ( ) [virtual]
```

Extract the next token from the input string.

Returns

The next unescaped character.

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

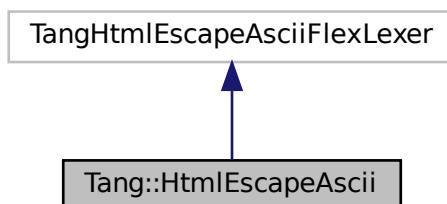
- [include/htmlEscape.hpp](#)

5.51 Tang::HtmlEscapeAscii Class Reference

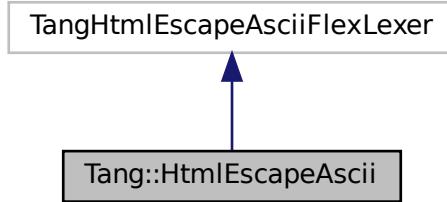
The Flex lexer class for the main Tang language.

```
#include <htmlEscapeAscii.hpp>
```

Inheritance diagram for Tang::HtmlEscapeAscii:



Collaboration diagram for Tang::HtmlEscapeAscii:



Public Member Functions

- [HtmlEscapeAscii \(std::istream &arg_yyin, std::ostream &arg_yyout, \[UnicodeString::Type type\\)\]\(#\)
The constructor for the Scanner.](#)
- [virtual std::string get_next_token \(\)](#)
Extract the next token from the input string.

Private Attributes

- [UnicodeString::Type type](#)
The type of string that is being escaped.

5.51.1 Detailed Description

The Flex lexer class for the main Tang language.

Flex requires that our lexer class inherit from yyFlexLexer, an "intermediate" class whose real name is "TangTangFlexLexer". We are subclassing it so that we can override the return type of [get_next_token\(\)](#), for compatibility with Bison 3 tokens.

5.51.2 Constructor & Destructor Documentation

5.51.2.1 HtmlEscapeAscii()

```
Tang::HtmlEscapeAscii::HtmlEscapeAscii (
    std::istream & arg_yyin,
    std::ostream & arg_yyout,
    UnicodeString::Type type ) [inline]
```

The constructor for the Scanner.

The design of the Flex lexer is to tokenize the contents of an input stream, and to write any error messages to an output stream. In our implementation, however, errors are returned differently, so the output stream is never used. Its presence is retained, however, in case it is needed in the future.

For now, the general approach should be to supply the input as a string stream, and to use std::cout as the output.

Parameters

<i>arg_yyin</i>	The input stream to be tokenized
<i>arg_yyout</i>	The output stream (not currently used)

5.51.3 Member Function Documentation**5.51.3.1 `get_next_token()`**

```
virtual std::string Tang::HtmlEscapeAscii::get_next_token ( ) [virtual]
```

Extract the next token from the input string.

Returns

The next unescaped character.

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

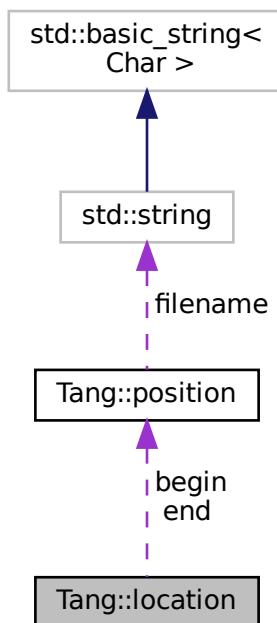
- [include/htmlEscapeAscii.hpp](#)

5.52 Tang::location Class Reference

Two points in a source file.

```
#include <location.hh>
```

Collaboration diagram for Tang::location:



Public Types

- `typedef position::filename_type filename_type`
Type for file name.
- `typedef position::counter_type counter_type`
Type for line and column numbers.

Public Member Functions

- `location (const position &b, const position &e)`
Construct a location from b to e.
- `location (const position &p=position())`
Construct a 0-width location in p.
- `location (filename_type *f, counter_type l=1, counter_type c=1)`
Construct a 0-width location in f, l, c.
- `void initialize (filename_type *f=((void *) 0), counter_type l=1, counter_type c=1)`
Initialization.

Line and Column related manipulators

- `void step ()`
Reset initial location to final location.
- `void columns (counter_type count=1)`
Extend the current location to the COUNT next columns.
- `void lines (counter_type count=1)`
Extend the current location to the COUNT next lines.

Public Attributes

- `position begin`
Beginning of the located region.
- `position end`
End of the located region.

5.52.1 Detailed Description

Two points in a source file.

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

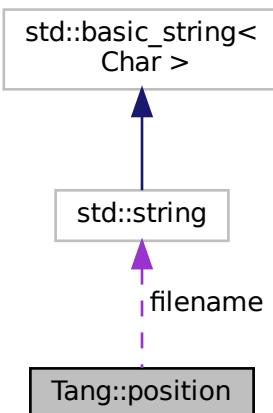
- build/generated/location.hh

5.53 Tang::position Class Reference

A point in a source file.

```
#include <location.hh>
```

Collaboration diagram for Tang::position:



Public Types

- `typedef const std::string filename_type`
Type for file name.
 - `typedef int counter_type`
Type for line and column numbers.

Public Member Functions

- **position** (*filename_type* $\ast f = (\text{void} \ast) 0$), **counter_type** $l = 1$, **counter_type** $c = 1$
Construct a position.
 - **void initialize** (*filename_type* $\ast fn = (\text{void} \ast) 0$), **counter_type** $l = 1$, **counter_type** $c = 1$)
Initialization.

Line and Column related manipulators

- void **lines** (**counter_type** count=1)
(line related) Advance to the COUNT next lines.
 - void **columns** (**counter_type** count=1)
(column related) Advance to the COUNT next columns.

Public Attributes

- `filename_type * filename`
`File name to which this position refers.`
- `counter_type line`
`Current line number.`
- `counter_type column`
`Current column number.`

Static Private Member Functions

- static `counter_type add_ (counter_type lhs, counter_type rhs, counter_type min)`
`Compute max (min, lhs+rhs).`

5.53.1 Detailed Description

A point in a source file.

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

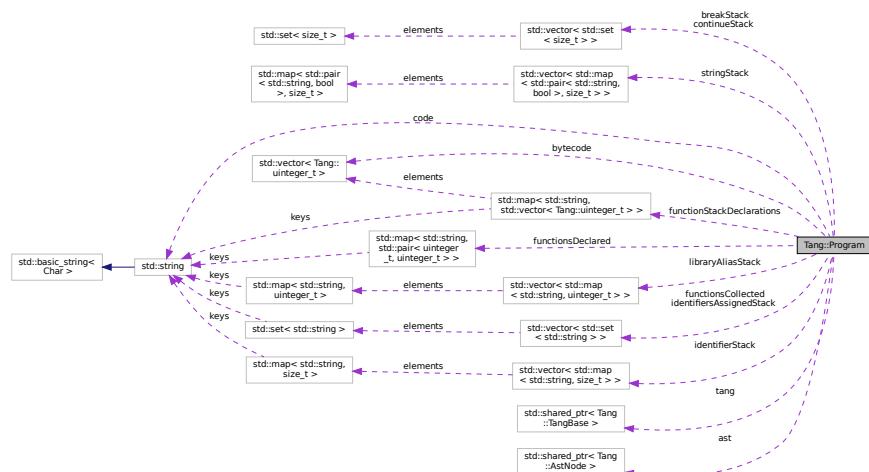
- `build/generated/location.hpp`

5.54 Tang::Program Class Reference

Represents a compiled script or template that may be executed.

```
#include <program.hpp>
```

Collaboration diagram for Tang::Program:



Public Types

- enum `CodeType` { `Script` , `Template` }
- Indicate the type of code that was supplied to the `Program`.*

Public Member Functions

- `Program` (std::string `code`, `CodeType codeType`, std::shared_ptr< `Tang::TangBase` > `tang`)
Create a compiled program using the provided code.
- std::string `getCode` () const
Get the code that was provided when the `Program` was created.
- std::optional< const std::shared_ptr< `AstNode` > > `getAst` () const
Get the AST that was generated by the parser.
- std::string `dumpBytecode` () const
Get the OpCodes of the compiled program, formatted like Assembly.
- std::optional< const `GarbageCollected` > `getResult` () const
Get the result of the `Program` execution, if it exists.
- size_t `addBytecode` (`Tang::uinteger_t`)
Add a `Tang::uinteger_t` to the Bytecode.
- const `Bytecode` & `getBytecode` ()
Get the Bytecode vector.
- `Context` `execute` ()
Execute the program's Bytecode, and return the execution `Context`.
- `Context` `execute` (`ContextData` &&`data`)
Execute the program's Bytecode, and return the execution `Context`.
- bool `setJumpTarget` (size_t `opcodeAddress`, `Tang::uinteger_t` `jumpTarget`)
Set the target address of a Jump opcode.
- bool `setFunctionStackDeclaration` (size_t `opcodeAddress`, `uinteger_t` `argc`, `uinteger_t` `targetPC`)
Set the stack details of a function declaration.
- void `pushEnvironment` (const std::shared_ptr< `AstNode` > &`ast`)
Create a new compile/execute environment stack entry.
- void `popEnvironment` ()
Remove a compile/execute environment stack entry.
- void `addIdentifier` (const std::string &`name`, std::optional< size_t > `position`={})
Add an identifier to the environment.
- const std::map< std::string, size_t > & `getIdentifiers` () const
Get the identifier map of the current environment.
- void `addLibraryAlias` (const std::string &`name`)
Add a library alias to the environment.
- const std::map< std::string, uinteger_t > & `getLibraryAliases` () const
Get the library alias map of the current environment.
- void `addIdentifierAssigned` (const std::string &`name`)
Indicate that an identifier will be altered within the associated scope.
- const std::set< std::string > & `getIdentifiersAssigned` () const
Get the set of identifiers that will be assigned in the current scope.
- void `addString` (const std::string &`name`, bool `isTrusted`)
Add a string to the environment.
- const std::map< std::pair< std::string, bool >, size_t > & `getStrings` () const
Get the string map of the current environment.
- void `pushBreakStack` ()

- **void addBreak (size_t location)**

Add the Bytecode location of a `break` statement, to be set when the final target is known at a later time.
- **void popBreakStack (size_t target)**

For all `continue` bytecode locations collected by `Tang::addContinue`, set the target pc to `target`.
- **void pushContinueStack ()**

Increase the `continue` environment stack, so that we can handle nested `continue`-supporting structures.
- **void addContinue (size_t location)**

Add the Bytecode location of a `continue` statement, to be set when the final target is known at a later time.
- **void popContinueStack (size_t target)**

For all `continue` bytecode locations collected by `Tang::addContinue`, set the target pc to `target`.

Public Attributes

- **std::vector< std::set< std::string > > functionsCollected**

Names of the functions that are declared in a previous or the current scope.
- **std::map< std::string, std::pair< uinteger_t, uinteger_t > > functionsDeclared**

Key/value pair of the function declaration information.
- **std::map< std::string, std::vector< Tang::uinteger_t > > functionStackDeclarations**

For each function name, a list of Bytecode addresses that need to be replaced by a function definition.

Private Member Functions

- **void parse ()**

Parse the code into an AST.
- **void compile ()**

Compile the AST into Bytecode.

Private Attributes

- **std::shared_ptr< Tang::TangBase > tang**

A pointer to the base `Tang` class.
- **std::vector< std::map< std::string, size_t > > identifierStack**

Stack of mappings of identifiers to their stack locations.
- **std::vector< std::map< std::string, uinteger_t > > libraryAliasStack**

Stack of library aliases that are used in the program.
- **std::vector< std::set< std::string > > identifiersAssignedStack**

Stack of sets of identifiers that are the target of an assignment statement within the associated scope.
- **std::vector< std::map< std::pair< std::string, bool >, size_t > > stringStack**

Stack of mappings of strings to their stack locations.
- **std::vector< std::set< size_t > > breakStack**

Stack of a collection of `break` statement locations.
- **std::vector< std::set< size_t > > continueStack**

Stack of a collection of `continue` statement locations.
- **std::string code**

The code supplied when the `Program` was instantiated.
- **CodeType codeType**

The type of code that was supplied when the `Program` was instantiated.
- **std::shared_ptr< AstNode > ast**

A pointer to the AST, if parsing was successful.
- **Bytecode bytecode**

The Bytecode of the compiled program.
- **std::optional< GarbageCollected > result**

The result of the `Program` compilation.

5.54.1 Detailed Description

Represents a compiled script or template that may be executed.

5.54.2 Member Enumeration Documentation

5.54.2.1 CodeType

enum `Tang::Program::CodeType`

Indicate the type of code that was supplied to the [Program](#).

Enumerator

Script	The code is pure Tang script, without any templating.
Template	The code is a template.

5.54.3 Constructor & Destructor Documentation

5.54.3.1 Program()

```
Program::Program (
    std::string code,
    Program::CodeType codeType,
    std::shared_ptr< Tang::TangBase > tang )
```

Create a compiled program using the provided code.

Parameters

<i>code</i>	The code to be compiled.
<i>codeType</i>	Whether the code is a Script or Template.
<i>tang</i>	A pointer to the base Tang class.

5.54.4 Member Function Documentation

5.54.4.1 addBreak()

```
void Program::addBreak (
    size_t location )
```

Add the Bytecode location of a `break` statement, to be set when the final target is known at a later time.

Parameters

<i>location</i>	The offset location of the <code>break</code> bytecode.
-----------------	---

5.54.4.2 addBytecode()

```
size_t Program::addBytecode (
    Tang::uinteger_t op )
```

Add a `Tang::uinteger_t` to the Bytecode.

Parameters

<i>op</i>	The value to add to the Bytecode.
-----------	-----------------------------------

Returns

The size of the bytecode structure.

5.54.4.3 addContinue()

```
void Program::addContinue (
    size_t location )
```

Add the Bytecode location of a `continue` statement, to be set when the final target is known at a later time.

Parameters

<i>location</i>	The offset location of the <code>continue</code> bytecode.
-----------------	--

5.54.4.4 addIdentifier()

```
void Program::addIdentifier (
    const std::string & name,
    std::optional< size_t > position = {} )
```

Add an identifier to the environment.

Parameters

<i>name</i>	The variable to add to the environment.
<i>position</i>	If provided, the desired position to place the identifier.

5.54.4.5 addIdentifierAssigned()

```
void Program::addIdentifierAssigned (
    const std::string & name )
```

Indicate that an identifier will be altered within the associated scope.

Parameters

<i>name</i>	The identifier name.
-------------	----------------------

5.54.4.6 addLibraryAlias()

```
void Program::addLibraryAlias (
    const std::string & name )
```

Add a library alias to the environment.

Parameters

<i>name</i>	The library alias to add to the environment.
-------------	--

5.54.4.7 addString()

```
void Program::addString (
    const std::string & name,
    bool isTrusted )
```

Add a string to the environment.

Parameters

<i>name</i>	The variable to add to the environment.
<i>isTrusted</i>	Whether or not the string is a trusted literal.

5.54.4.8 dumpBytecode()

```
string Program::dumpBytecode ( ) const
```

Get the Opcodes of the compiled program, formatted like Assembly.

Returns

A string containing the Opcode representation.

5.54.4.9 execute() [1/2]

```
Context Program::execute ( )
```

Execute the program's Bytecode, and return the execution [Context](#).

A default ContextData will be generated for the execution.

Returns

The execution [Context](#).

5.54.4.10 execute() [2/2]

```
Context Program::execute ( ContextData && data )
```

Execute the program's Bytecode, and return the execution [Context](#).

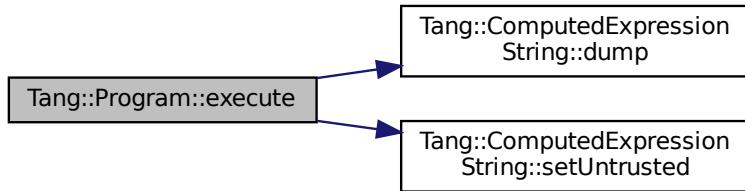
Parameters

<code>data</code>	The default data to be made available to the execution Context .
-------------------	--

Returns

The execution [Context](#).

Here is the call graph for this function:



5.54.4.11 `getAst()`

```
optional< const shared_ptr< AstNode > > Program::getAst ( ) const
```

Get the AST that was generated by the parser.

The parser may have failed, so the return is an `optional<>` type. If the compilation failed, check `Program::error`.

Returns

A pointer to the AST, if it exists.

5.54.4.12 `getBytecode()`

```
const Bytecode & Program::getBytecode ( )
```

Get the Bytecode vector.

Returns

The Bytecode vector.

5.54.4.13 getCode()

```
string Program::getCode ( ) const
```

Get the code that was provided when the [Program](#) was created.

Returns

The source code from which the [Program](#) was created.

5.54.4.14 getIdentifiers()

```
const map< string, size_t > & Program::getIdentifiers ( ) const
```

Get the identifier map of the current environment.

Returns

A map of each identifier name to its stack position within the current environment.

5.54.4.15 getIdentifiersAssigned()

```
const set< string > & Program::getIdentifiersAssigned ( ) const
```

Get the set of identifiers that will be assigned in the current scope.

Returns

A set of identifier names that have been identified as the target of an assignment operator within the current scope.

5.54.4.16 getLibraryAliases()

```
const map< string, uinteger_t > & Program::getLibraryAliases ( ) const
```

Get the library alias map of the current environment.

Returns

A map of each library alias to its stack position within the current environment.

5.54.4.17 getResult()

```
optional< const GarbageCollected > Program::getResult ( ) const
```

Get the result of the [Program](#) execution, if it exists.

Returns

The result of the [Program](#) execution, if it exists.

5.54.4.18 getStrings()

```
const map< pair< string, bool >, size_t > & Program::getStrings ( ) const
```

Get the string map of the current environment.

Returns

A map of each identifier name to its stack position within the current environment.

5.54.4.19 popBreakStack()

```
void Program::popBreakStack ( size_t target )
```

For all `continue` bytecode locations collected by `Tang::addContinue`, set the target pc to `target`.

Parameters

<code>target</code>	The target bytecode offset that the <code>continue</code> should jump to.
---------------------	---

Here is the call graph for this function:



5.54.4.20 popContinueStack()

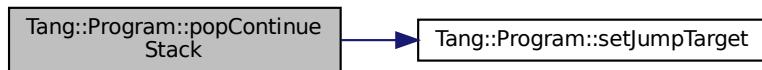
```
void Program::popContinueStack ( size_t target )
```

For all `continue` bytecode locations collected by `Tang::addContinue`, set the target pc to `target`.

Parameters

<code>target</code>	The target bytecode offset that the <code>continue</code> should jump to.
---------------------	---

Here is the call graph for this function:



5.54.4.21 pushEnvironment()

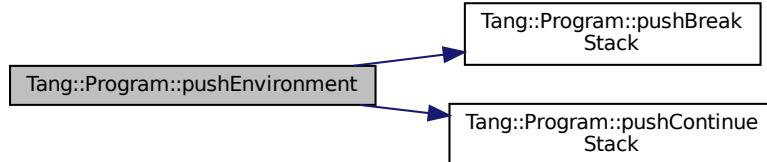
```
void Program::pushEnvironment ( const std::shared_ptr< AstNode > & ast )
```

Create a new compile/execute environment stack entry.

Parameters

<code>ast</code>	The ast node from which this new environment will be formed.
------------------	--

Here is the call graph for this function:



5.54.4.22 `setFunctionStackDeclaration()`

```
bool Program::setFunctionStackDeclaration (
    size_t opcodeAddress,
    uinteger_t argc,
    uinteger_t targetPC )
```

Set the stack details of a function declaration.

Parameters

<i>opcodeAddress</i>	The location of the FUNCTION opcode.
<i>argc</i>	The argument count to set.
<i>targetPC</i>	The bytecode address of the start of the function.

5.54.4.23 `setJumpTarget()`

```
bool Program::setJumpTarget (
    size_t opcodeAddress,
    Tang::uinteger_t jumpTarget )
```

Set the target address of a Jump opcode.

Parameters

<i>opcodeAddress</i>	The location of the jump statement.
<i>jumpTarget</i>	The address to jump to.

Returns

Whether or not the jumpTarget was set.

5.54.5 Member Data Documentation

5.54.5.1 `functionsDeclared`

```
std::map<std::string, std::pair<uinteger_t, uinteger_t> > Tang::Program::functionsDeclared
```

Key/value pair of the function declaration information.

The key is the name of the function. The value is a pair of the argc value and the targetPC value.

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

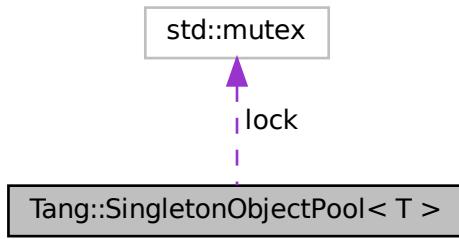
- [include/program.hpp](#)
- [src/program-dumpBytecode.cpp](#)
- [src/program-execute.cpp](#)
- [src/program.cpp](#)

5.55 Tang::SingletonObjectPool< T > Class Template Reference

A thread-safe, singleton object pool of the designated type.

```
#include <singletonObjectPool.hpp>
```

Collaboration diagram for Tang::SingletonObjectPool< T >:



Public Member Functions

- `T * get ()`
Request an uninitialized memory location from the pool for an object T.
- `void recycle (T *obj)`
Recycle a memory location for an object T.
- `~SingletonObjectPool ()`
Destructor.

Static Public Member Functions

- `static SingletonObjectPool< T > & getInstance ()`
Get the singleton instance of the object pool.

Private Member Functions

- `SingletonObjectPool ()`
The constructor, hidden from being directly called.
- `SingletonObjectPool (const SingletonObjectPool &other)`
The copy constructor, hidden from being called.

Private Attributes

- `T ** allocations`
C-array of allocated blocks, each block contains GROW objects.
- `int currentAllocation`
Index into allocations, representing the current block supplying non-recycled memory addresses.
- `size_t currentIndex`
Current location (within the most recently allocated block) of an available T.*
- `int currentRecycledAllocation`
Index into allocations, representing the current block tracking the recycled memory addresses.
- `int currentRecycledIndex`
Current location (within the currentRecycledAllocation block) of the last available T.*

Static Private Attributes

- `static std::mutex lock`
A mutex for thread-safety.

5.55.1 Detailed Description

```
template<class T>
class Tang::SingletonObjectPool< T >
```

A thread-safe, singleton object pool of the designated type.

5.55.2 Member Function Documentation

5.55.2.1 get()

```
template<class T >
T* Tang::SingletonObjectPool< T >::get ( ) [inline]
```

Request an uninitialized memory location from the pool for an object T.

Returns

An uninitialized memory location for an object T.

5.55.2.2 getInstance()

```
template<class T >
static SingletonObjectPool<T>& Tang::SingletonObjectPool< T >::getInstance ( ) [inline],
[static]
```

Get the singleton instance of the object pool.

Returns

The singleton instance of the object pool.

5.55.2.3 recycle()

```
template<class T >
void Tang::SingletonObjectPool< T >::recycle (
    T * obj ) [inline]
```

Recycle a memory location for an object T.

Parameters

<i>obj</i>	The memory location to recycle.
------------	---------------------------------

5.55.3 Member Data Documentation

5.55.3.1 currentIndex

```
template<class T >
size_t Tang::SingletonObjectPool< T >::currentIndex [private]
```

Current location (within the most recently allocated block) of an available T*.

If currentIndex == GROW, then a new block needs to be allocated.

5.55.3.2 currentRecycledIndex

```
template<class T >
int Tang::SingletonObjectPool< T >::currentRecycledIndex [private]
```

Current location (within the currentRecycledAllocation block) of the last available T*.

If currentRecycledIndex == GROW, then we must move to the next currentRecycledAllocation.

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

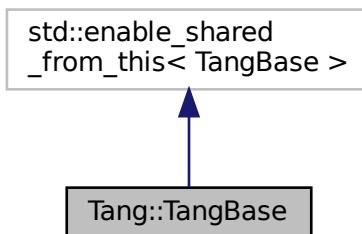
- [include/SingletonObjectPool.hpp](#)

5.56 Tang::TangBase Class Reference

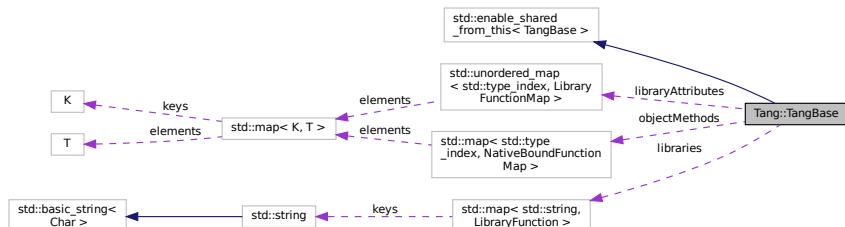
The base class for the Tang programming language.

```
#include <tangBase.hpp>
```

Inheritance diagram for Tang::TangBase:



Collaboration diagram for Tang::TangBase:



Public Member Functions

- **Program compileScript (std::string script)**
Compile the provided text as a script and return a [Program](#).
- **Program compileTemplate (std::string code)**
Compile the provided text as a template and return a [Program](#).
- **TangBase ()**
The constructor.
- **std::map< std::type_index, NativeBoundFunctionMap > & getObjectMethods ()**
Get the object methods available to this instance of the base language object.
- **LibraryFunctionMap & getLibraries ()**
Get the libraries available to this instance of the base language object.
- **std::unordered_map< std::type_index, LibraryFunctionMap > & getLibraryAttributes ()**
Get the library attributes available to this instance of the base language object.

Static Public Member Functions

- static std::shared_ptr< [TangBase](#) > [make_shared](#) ()

Create an instance of Tang and return a reference to it as a shared pointer.

Private Attributes

- std::map< std::type_index, [NativeBoundFunctionMap](#) > [objectMethods](#)

Store the available object methods.
- [LibraryFunctionMap](#) [libraries](#)

Store the available libraries.
- std::unordered_map< std::type_index, [LibraryFunctionMap](#) > [libraryAttributes](#)

Store the available library attributes.

5.56.1 Detailed Description

The base class for the Tang programming language.

This class is the fundamental starting point to compile and execute a Tang program. It may be considered in three parts:

1. It acts as an extendable interface through which additional "library" functions can be added to the language. It is intentionally designed that each instance of [TangBase](#) will have its own library functions.
2. It provides methods to compile scripts and templates, resulting in a [Program](#) object.
3. The [Program](#) object may then be executed, providing instance-specific context information (*i.e.*, state).

5.56.2 Constructor & Destructor Documentation

5.56.2.1 [TangBase\(\)](#)

```
TangBase::TangBase ( )
```

The constructor.

This function should never be called directly. Rather, always use the [Tang::TangBase\(\)](#) static method, which supplies the shared pointer necessary for creation of [Program](#) objects. Here is the call graph for this function:



5.56.3 Member Function Documentation

5.56.3.1 compileScript()

```
Program TangBase::compileScript (
    std::string script )
```

Compile the provided text as a script and return a [Program](#).

Parameters

<i>script</i>	The Tang script to be compiled.
---------------	---------------------------------

Returns

The [Program](#) object representing the compiled script.

5.56.3.2 compileTemplate()

```
Program TangBase::compileTemplate (
    std::string code )
```

Compile the provided text as a template and return a [Program](#).

Parameters

<i>code</i>	The Tang template to be compiled.
-------------	-----------------------------------

Returns

The [Program](#) object representing the compiled template.

5.56.3.3 make_shared()

```
shared_ptr< TangBase > TangBase::make_shared ( ) [static]
```

Create an instance of Tang and return a reference to it as a shared pointer.

Returns

A shared pointer to the base Tang object.

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

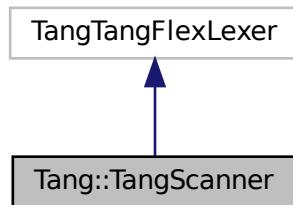
- [include/tangBase.hpp](#)
- [src/tangBase.cpp](#)

5.57 Tang::TangScanner Class Reference

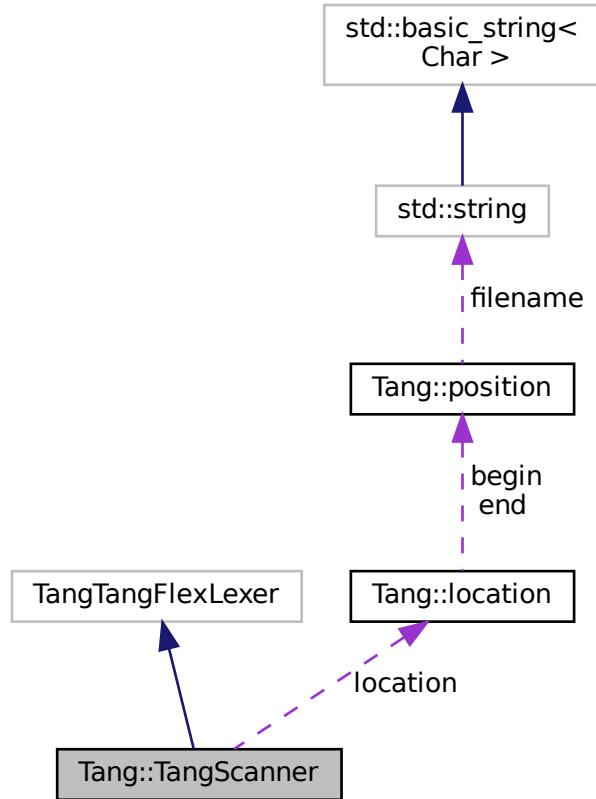
The Flex lexer class for the main Tang language.

```
#include <tangScanner.hpp>
```

Inheritance diagram for Tang::TangScanner:



Collaboration diagram for Tang::TangScanner:



Public Member Functions

- [TangScanner](#) (std::istream &arg_yyin, std::ostream &arg_yyout)
The constructor for the Scanner.
- virtual Tang::TangParser::symbol_type [get_next_token](#) ()
A pass-through function that we supply so that we can provide a Bison 3 token return type instead of the `int` that is returned by the default class configuration.
- void [setModeTemplate](#) ()
Helper function to set the scanner to template parsing mode.

Private Attributes

- [Tang::location](#) location
The location information of the token that is identified.

5.57.1 Detailed Description

The Flex lexer class for the main Tang language.

Flex requires that our lexer class inherit from yyFlexLexer, an "intermediate" class whose real name is "TangTangFlexLexer". We are subclassing it so that we can override the return type of [get_next_token\(\)](#), for compatibility with Bison 3 tokens.

5.57.2 Constructor & Destructor Documentation

5.57.2.1 [TangScanner\(\)](#)

```
Tang::TangScanner::TangScanner (
    std::istream & arg_yyin,
    std::ostream & arg_yyout ) [inline]
```

The constructor for the Scanner.

The design of the Flex lexer is to tokenize the contents of an input stream, and to write any error messages to an output stream. In our implementation, however, errors are returned differently, so the output stream is never used. Its presence is retained, however, in case it is needed in the future.

For now, the general approach should be to supply the input as a string stream, and to use `std::cout` as the output.

Parameters

<code>arg_yyin</code>	The input stream to be tokenized
<code>arg_yyout</code>	The output stream (not currently used)

5.57.3 Member Function Documentation

5.57.3.1 `get_next_token()`

```
virtual Tang::TangParser::symbol_type Tang::TangScanner::get_next_token ( ) [virtual]
```

A pass-through function that we supply so that we can provide a Bison 3 token return type instead of the `int` that is returned by the default class configuration.

Returns

A Bison 3 token representing the lexeme that was recognized.

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

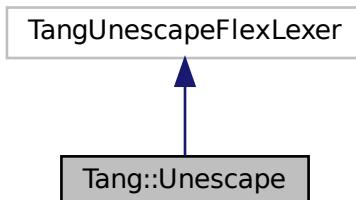
- [include/tangScanner.hpp](#)

5.58 Tang::Unescape Class Reference

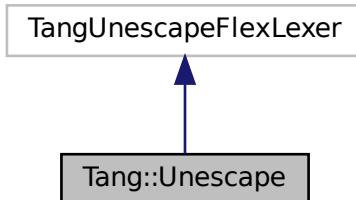
The Flex lexer class for the main Tang language.

```
#include <unescape.hpp>
```

Inheritance diagram for Tang::Unescape:



Collaboration diagram for Tang::Unescape:



Public Member Functions

- [Unescape](#) (std::istream &arg_yyin, std::ostream &arg_yyout)
The constructor for the Scanner.
- virtual std::string [get_next_token](#) ()
Extract the next token from the input string.

5.58.1 Detailed Description

The Flex lexer class for the main Tang language.

Flex requires that our lexer class inherit from yyFlexLexer, an "intermediate" class whose real name is "TangTang←FlexLexer". We are subclassing it so that we can override the return type of [get_next_token\(\)](#), for compatibility with Bison 3 tokens.

5.58.2 Constructor & Destructor Documentation

5.58.2.1 Unescape()

```
Tang::Unescape::Unescape (
    std::istream & arg_yyin,
    std::ostream & arg_yyout ) [inline]
```

The constructor for the Scanner.

The design of the Flex lexer is to tokenize the contents of an input stream, and to write any error messages to an output stream. In our implementation, however, errors are returned differently, so the output stream is never used. Its presence is retained, however, in case it is needed in the future.

For now, the general approach should be to supply the input as a string stream, and to use std::cout as the output.

Parameters

<i>arg_yyin</i>	The input stream to be tokenized
<i>arg_yyout</i>	The output stream (not currently used)

5.58.3 Member Function Documentation

5.58.3.1 get_next_token()

```
virtual std::string Tang::Unescape::get_next_token ( ) [virtual]
```

Extract the next token from the input string.

Returns

The next unescaped character.

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

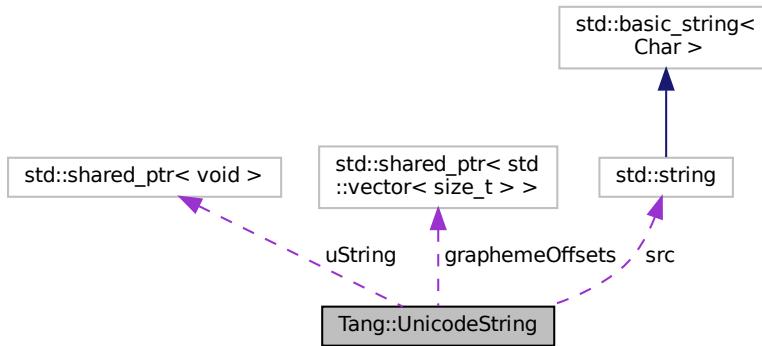
- [include/unescape.hpp](#)

5.59 `Tang::UnicodeString` Class Reference

Represents a UTF-8 encoded string that is Unicode-aware.

```
#include <unicodeString.hpp>
```

Collaboration diagram for `Tang::UnicodeString`:



Public Types

- enum `Type` { [Trusted](#) , [Untrusted](#) }

The types of string being created.

Public Member Functions

- [UnicodeString \(\)](#)
Construct an empty `Tang::UnicodeString` object, which acts as the interface to the ICU library.
- [UnicodeString \(const UnicodeString &source\)](#)
Construct a `Tang::UnicodeString` object, from an existing `Tang::UnicodeString`.
- [UnicodeString \(const std::string &source\)](#)
Construct a `Tang::UnicodeString` object, from an existing `std::string`.
- `std::string substr (size_t position, size_t length) const`
Return a Unicode grapheme-aware substring.
- `bool operator== (const UnicodeString &rhs) const`
Compare two `UnicodeString`s.

- `bool operator< (const UnicodeString &rhs) const`
`Compare two UnicodeStrings.`
- `UnicodeString operator+ (const UnicodeString &rhs) const`
`Create a new UnicodeString that is the concatenation of two UnicodeStrings.`
- `UnicodeString & operator+= (const UnicodeString &rhs)`
`Concatenate the rhs UnicodeString to the current UnicodeString.`
- `operator std::string () const`
`Cast the current UnicodeString object to a std::string, UTF-8 encoded.`
- `size_t length () const`
`Return the length of the UnicodeString in graphemes.`
- `size_t bytesLength () const`
`Return the length of the UnicodeString in bytes.`
- `std::string render () const`
`Render the string in with dangerous characters HTML encoded, if the string is UnicodeString::Type::Untrusted.`
- `std::string renderAscii () const`
`Render the string in with all characters converted to an ASCII representation.`
- `void setUntrusted ()`
`Set the string as UnicodeString::Type::Untrusted.`

Private Member Functions

- `void generateCachedValues () const`
`Calculate cachable values for the object.`

Private Attributes

- `std::string src`
`The UTF-8 encoded string.`
- `UnicodeString::Type type`
`The type of string being stored.`
- `std::shared_ptr< std::vector< size_t > > graphemeOffsets`
`Cache of the grapheme offsets, if they happen to be calculated.`
- `std::shared_ptr< void > uString`
`Cache of the ICU Unicode string.`

5.59.1 Detailed Description

Represents a UTF-8 encoded string that is Unicode-aware.

This class serves as the interface between the Tang language and the ICU library.

5.59.2 Member Enumeration Documentation

5.59.2.1 Type

```
enum Tang::UnicodeString::Type
```

The types of string being created.

Enumerator

Trusted	String is from a trusted source.
Untrusted	String is not from a trusted source.

5.59.3 Member Function Documentation

5.59.3.1 bytesLength()

```
size_t UnicodeString::bytesLength ( ) const
```

Return the length of the [UnicodeString](#) in bytes.

Note: this is *not* the number of codepoints or graphemes, but is the actual number of bytes in memory.

Returns

Returns the length of the [UnicodeString](#) in bytes.

5.59.3.2 length()

```
size_t UnicodeString::length ( ) const
```

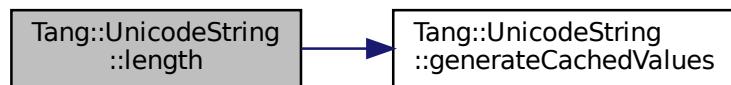
Return the length of the [UnicodeString](#) in graphemes.

Note: this is *not* the number of bytes, chars, or codepoints, but is the length in graphemes, as defined by ICU.

Returns

Returns the length of the [UnicodeString](#) in graphemes.

Here is the call graph for this function:



5.59.3.3 operator std::string()

```
UnicodeString::operator std::string ( ) const
```

Cast the current [UnicodeString](#) object to a std::string, UTF-8 encoded.

Returns

Returns the std::string version of the [UnicodeString](#).

5.59.3.4 operator+()

```
UnicodeString UnicodeString::operator+ (  
    const UnicodeString & rhs ) const
```

Create a new [UnicodeString](#) that is the concatenation of two UnicodeStrings.

Parameters

<i>rhs</i>	The string to append to the current object string.
------------	--

Returns

Returns the result of the concatenation.

5.59.3.5 operator+=()

```
UnicodeString & UnicodeString::operator+= (   
    const UnicodeString & rhs )
```

Concatenate the *rhs* [UnicodeString](#) to the current [UnicodeString](#).

Parameters

<i>rhs</i>	The string to append to the current object string.
------------	--

Returns

Returns the result of the concatenation.

5.59.3.6 operator<()

```
bool UnicodeString::operator< (
    const UnicodeString & rhs ) const
```

Compare two UnicodeStrings.

Parameters

<i>rhs</i>	The string to compare against.
------------	--------------------------------

Returns

Returns true if the rhs string is greater than or equal to the object string.

5.59.3.7 operator==()

```
bool UnicodeString::operator== (
    const UnicodeString & rhs ) const
```

Compare two UnicodeStrings.

Parameters

<i>rhs</i>	The string to compare against.
------------	--------------------------------

Returns

Returns true if the two strings are equal.

5.59.3.8 render()

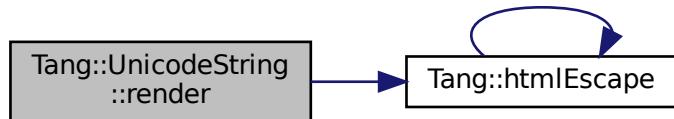
```
string UnicodeString::render ( ) const
```

Render the string in with dangerous characters HTML encoded, if the string is UnicodeString::Type::Untrusted.

Returns

The rendered string, according to its type.

Here is the call graph for this function:

**5.59.3.9 renderAscii()**

```
string UnicodeString::renderAscii ( ) const
```

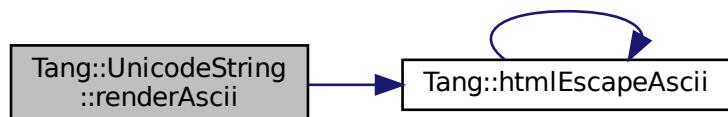
Render the string in with all characters converted to an ASCII representation.

The dangerous characters will not be HTML encoded, if the string is `UnicodeString::Type::Trusted`.

Returns

The rendered string, according to its type.

Here is the call graph for this function:

**5.59.3.10 substr()**

```
std::string UnicodeString::substr ( size_t position, size_t length ) const
```

Return a Unicode grapheme-aware substring.

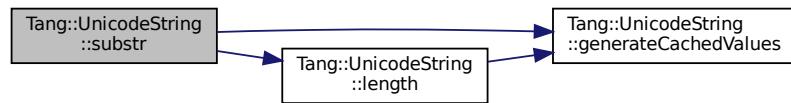
Parameters

<i>position</i>	The 0-based position of the first grapheme.
<i>length</i>	The maximum number of graphemes to return.

Returns

The requested substring.

Here is the call graph for this function:



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

- [include/unicodeString.hpp](#)
- [src/unicodeString.cpp](#)

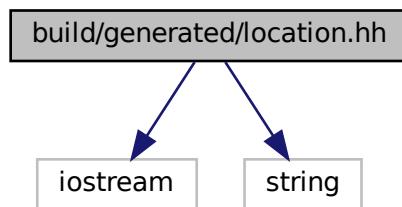
Chapter 6

File Documentation

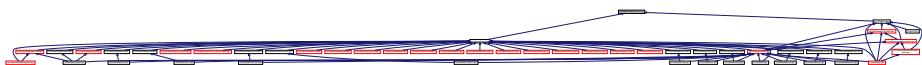
6.1 build/generated/location.hh File Reference

Define the Tang ::location class.

```
#include <iostream>
#include <string>
Include dependency graph for location.hh:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::position](#)
A point in a source file.
- class [Tang::location](#)
Two points in a source file.

Macros

- `#define YY_NULLPTR ((void*)0)`

Functions

- `position & Tang::operator+= (position &res, position::counter_type width)`
Add width columns, in place.
- `position Tang::operator+ (position res, position::counter_type width)`
Add width columns.
- `position & Tang::operator-= (position &res, position::counter_type width)`
Subtract width columns, in place.
- `position Tang::operator- (position res, position::counter_type width)`
Subtract width columns.
- `template<typename YYChar >`
`std::basic_ostream< YYChar > & Tang::operator<< (std::basic_ostream< YYChar > &ostr, const position &pos)`
Intercept output stream redirection.
- `location & Tang::operator+= (location &res, const location &end)`
Join two locations, in place.
- `location Tang::operator+ (location res, const location &end)`
Join two locations.
- `location & Tang::operator+= (location &res, location::counter_type width)`
Add width columns to the end position, in place.
- `location Tang::operator+ (location res, location::counter_type width)`
Add width columns to the end position.
- `location & Tang::operator-= (location &res, location::counter_type width)`
Subtract width columns to the end position, in place.
- `location Tang::operator- (location res, location::counter_type width)`
Subtract width columns to the end position.
- `template<typename YYChar >`
`std::basic_ostream< YYChar > & Tang::operator<< (std::basic_ostream< YYChar > &ostr, const location &loc)`
Intercept output stream redirection.

6.1.1 Detailed Description

Define the `Tang ::location` class.

6.1.2 Function Documentation

6.1.2.1 operator<<() [1/2]

```
template<typename YYChar >
std::basic_ostream<YYChar>& Tang::operator<< (
    std::basic_ostream< YYChar > & ostr,
    const location & loc )
```

Intercept output stream redirection.

Parameters

<i>ostr</i>	the destination output stream
<i>loc</i>	a reference to the location to redirect

Avoid duplicate information.

6.1.2.2 operator<<() [2/2]

```
template<typename YYChar >
std::basic_ostream<YYChar>& Tang::operator<< (
    std::basic_ostream< YYChar > & ostr,
    const position & pos )
```

Intercept output stream redirection.

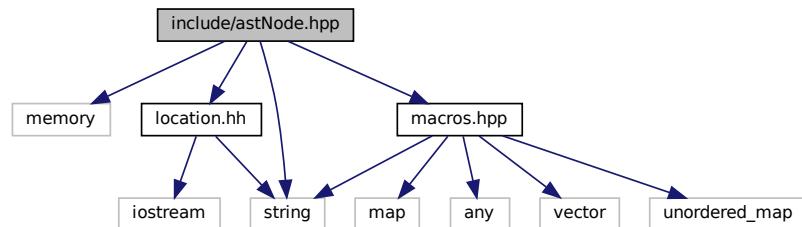
Parameters

<i>ostr</i>	the destination output stream
<i>pos</i>	a reference to the position to redirect

6.2 include/astNode.hpp File Reference

Declare the [Tang::AstNode](#) base class.

```
#include <memory>
#include <string>
#include "location.hh"
#include "macros.hpp"
Include dependency graph for astNode.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNode](#)
Base class for representing nodes of an Abstract Syntax Tree (AST).

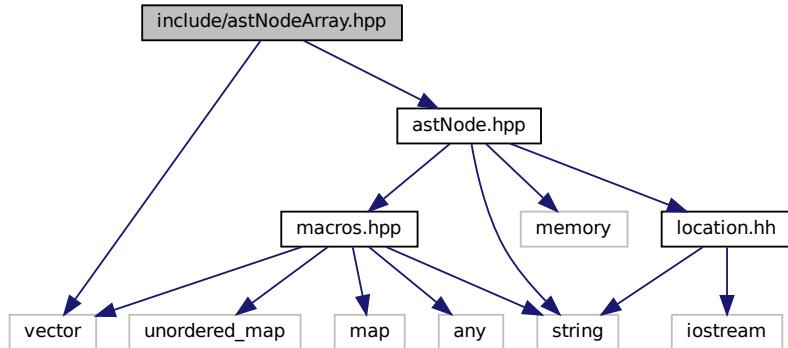
6.2.1 Detailed Description

Declare the [Tang::AstNode](#) base class.

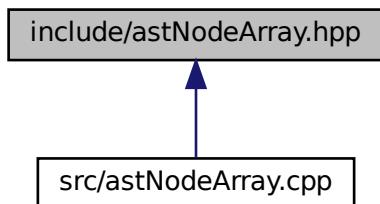
6.3 include/astNodeArray.hpp File Reference

Declare the [Tang::AstNodeArray](#) class.

```
#include <vector>
#include "astNode.hpp"
Include dependency graph for astNodeArray.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeArray](#)
An [AstNode](#) that represents an array literal.

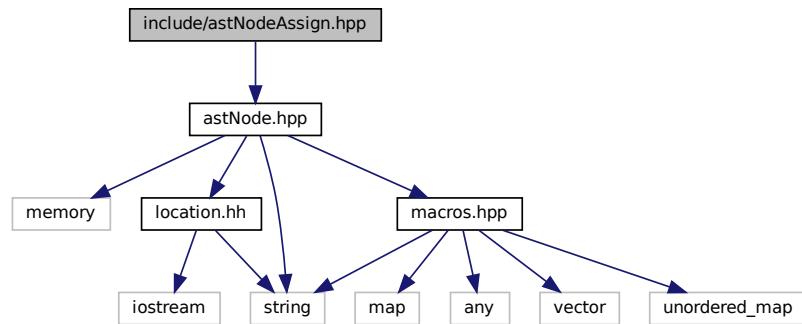
6.3.1 Detailed Description

Declare the [Tang::AstNodeArray](#) class.

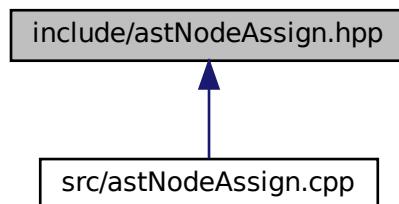
6.4 include/astNodeAssign.hpp File Reference

Declare the [Tang::AstNodeAssign](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeAssign.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeAssign](#)
An [AstNode](#) that represents a binary expression.

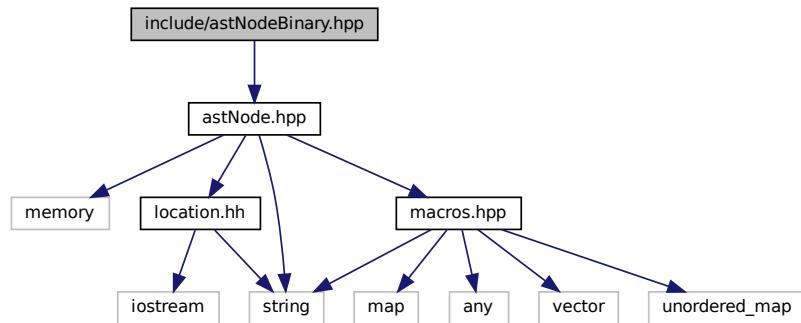
6.4.1 Detailed Description

Declare the [Tang::AstNodeAssign](#) class.

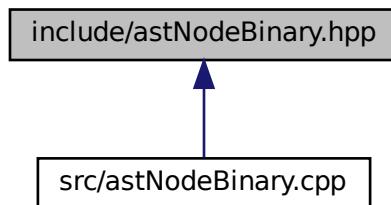
6.5 include/astNodeBinary.hpp File Reference

Declare the [Tang::AstNodeBinary](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeBinary.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeBinary](#)
An `AstNode` that represents a binary expression.

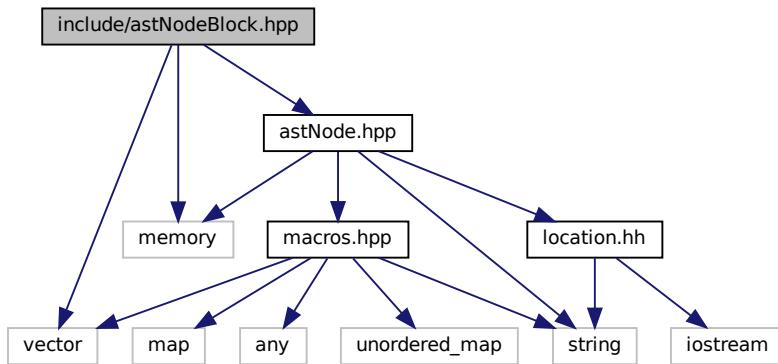
6.5.1 Detailed Description

Declare the [Tang::AstNodeBinary](#) class.

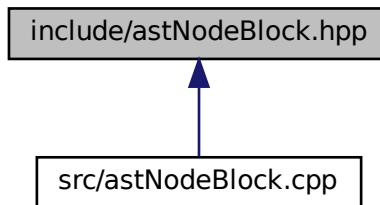
6.6 include/astNodeBlock.hpp File Reference

Declare the [Tang::AstNodeBlock](#) class.

```
#include <vector>
#include <memory>
#include "astNode.hpp"
Include dependency graph for astNodeBlock.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeBlock](#)
An [AstNode](#) that represents a code block.

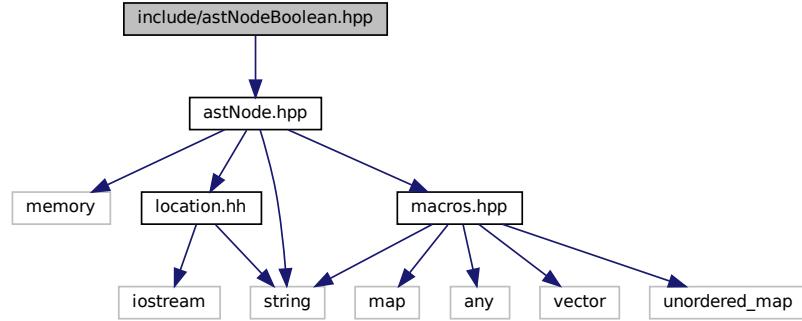
6.6.1 Detailed Description

Declare the [Tang::AstNodeBlock](#) class.

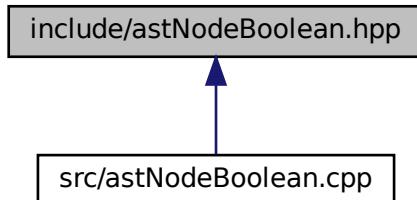
6.7 include/astNodeBoolean.hpp File Reference

Declare the [Tang::AstNodeBoolean](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeBoolean.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeBoolean](#)
An `AstNode` that represents a boolean literal.

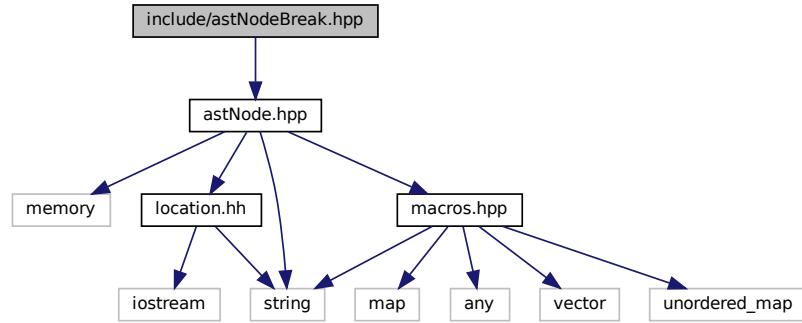
6.7.1 Detailed Description

Declare the [Tang::AstNodeBoolean](#) class.

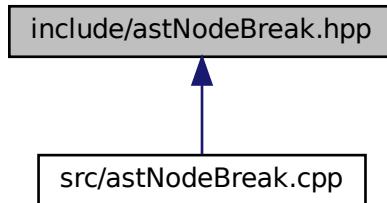
6.8 include/astNodeBreak.hpp File Reference

Declare the [Tang::AstNodeBreak](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeBreak.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeBreak](#)
An `AstNode` that represents a `break` statement.

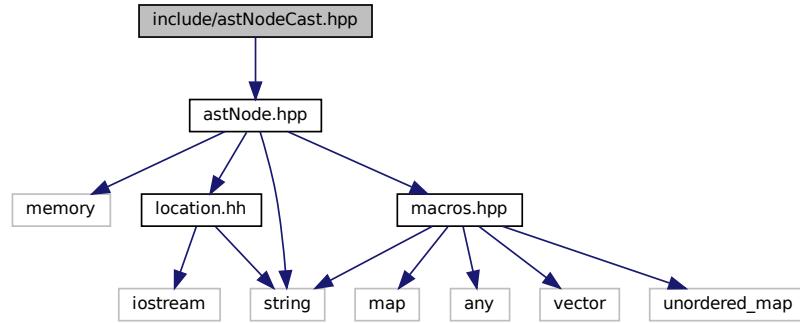
6.8.1 Detailed Description

Declare the [Tang::AstNodeBreak](#) class.

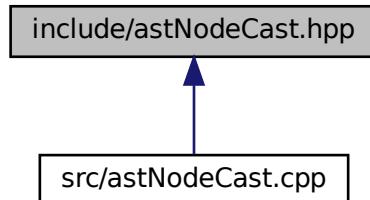
6.9 include/astNodeCast.hpp File Reference

Declare the [Tang::AstNodeCast](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeCast.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeCast](#)
An `AstNode` that represents a typecast of an expression.

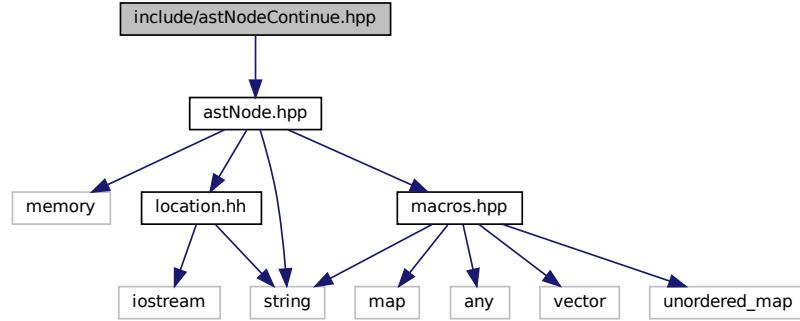
6.9.1 Detailed Description

Declare the [Tang::AstNodeCast](#) class.

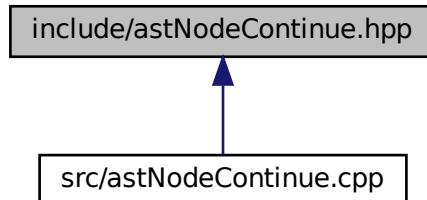
6.10 include/astNodeContinue.hpp File Reference

Declare the [Tang::AstNodeContinue](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeContinue.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeContinue](#)
An `AstNode` that represents a `continue` statement.

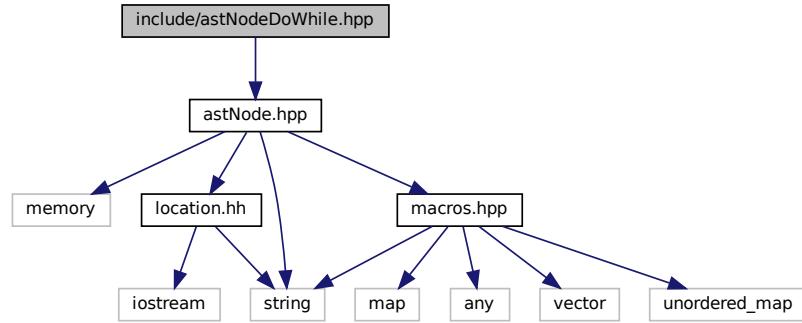
6.10.1 Detailed Description

Declare the [Tang::AstNodeContinue](#) class.

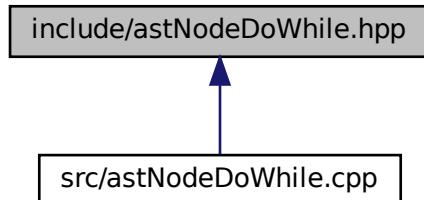
6.11 include/astNodeDoWhile.hpp File Reference

Declare the [Tang::AstNodeDoWhile](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeDoWhile.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeDoWhile](#)
An `AstNode` that represents a do..while statement.

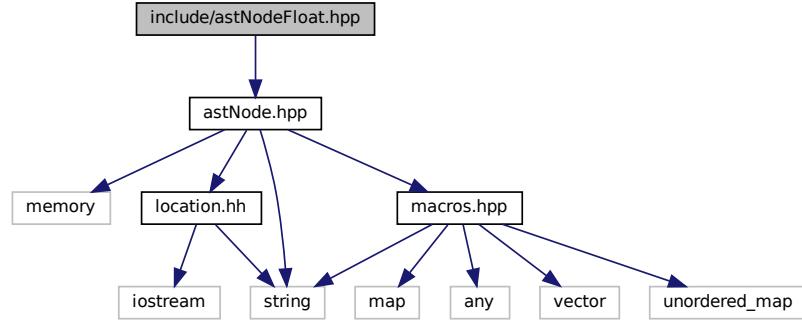
6.11.1 Detailed Description

Declare the [Tang::AstNodeDoWhile](#) class.

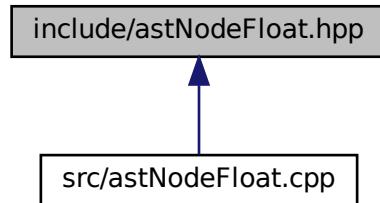
6.12 include/astNodeFloat.hpp File Reference

Declare the [Tang::AstNodeFloat](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeFloat.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeFloat](#)
An `AstNode` that represents an float literal.

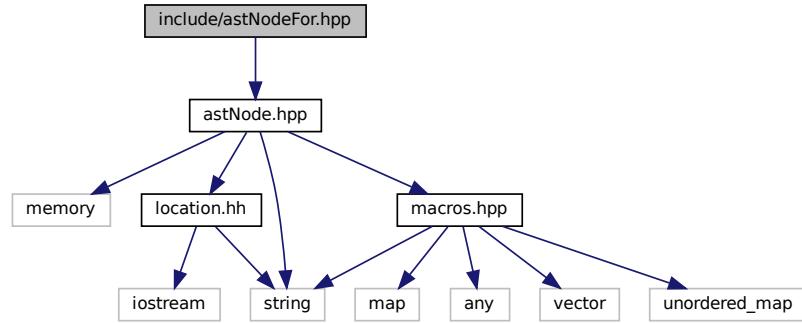
6.12.1 Detailed Description

Declare the [Tang::AstNodeFloat](#) class.

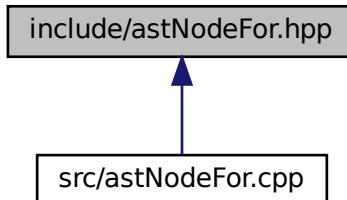
6.13 include/astNodeFor.hpp File Reference

Declare the [Tang::AstNodeFor](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeFor.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeFor](#)
An `AstNode` that represents an `if()` statement.

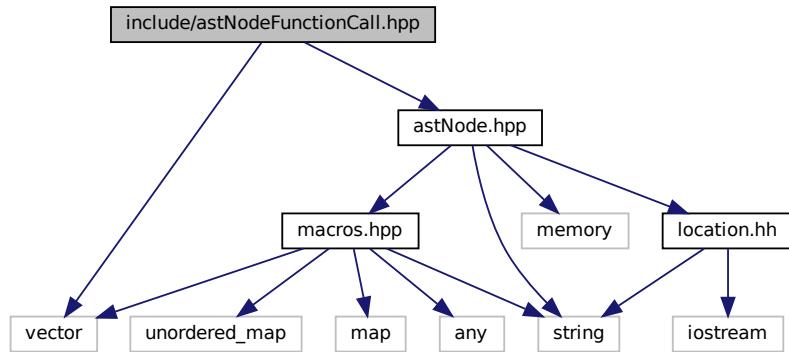
6.13.1 Detailed Description

Declare the [Tang::AstNodeFor](#) class.

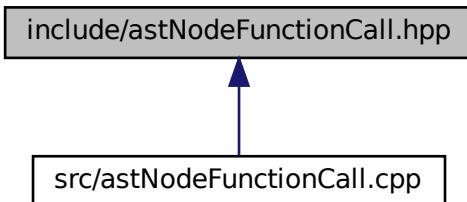
6.14 include/astNodeFunctionCall.hpp File Reference

Declare the [Tang::AstNodeFunctionCall](#) class.

```
#include <vector>
#include "astNode.hpp"
Include dependency graph for astNodeFunctionCall.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeFunctionCall](#)
An [AstNode](#) that represents a function call.

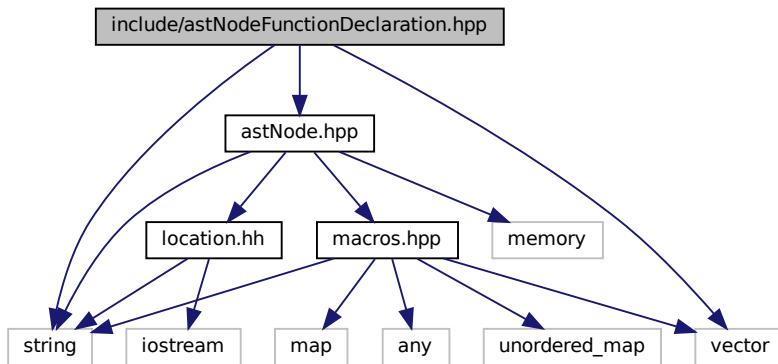
6.14.1 Detailed Description

Declare the [Tang::AstNodeFunctionCall](#) class.

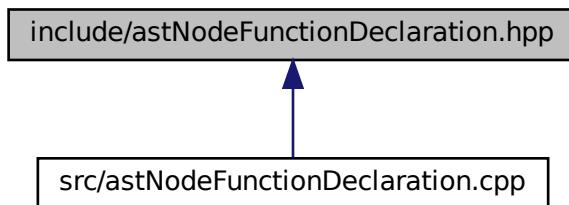
6.15 include/astNodeFunctionDeclaration.hpp File Reference

Declare the [Tang::AstNodeFunctionDeclaration](#) class.

```
#include <string>
#include <vector>
#include "astNode.hpp"
Include dependency graph for astNodeFunctionDeclaration.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeFunctionDeclaration](#)
An [AstNode](#) that represents a function declaration.

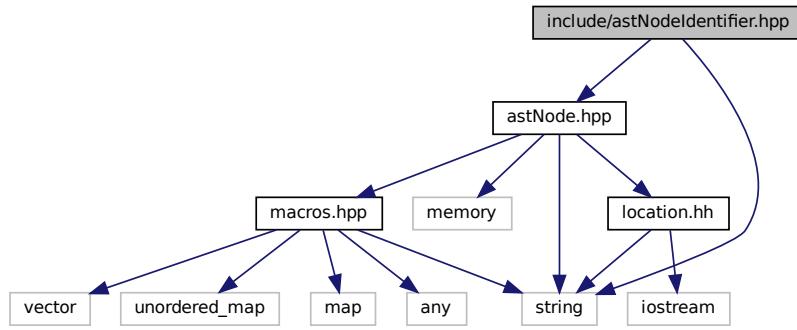
6.15.1 Detailed Description

Declare the [Tang::AstNodeFunctionDeclaration](#) class.

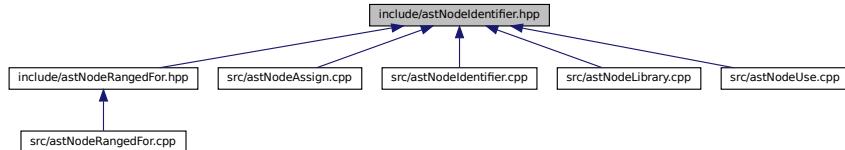
6.16 include/astNodelentifier.hpp File Reference

Declare the [Tang::AstNodelentifier](#) class.

```
#include <string>
#include "astNode.hpp"
Include dependency graph for astNodelentifier.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodelentifier](#)
An `AstNode` that represents an identifier.

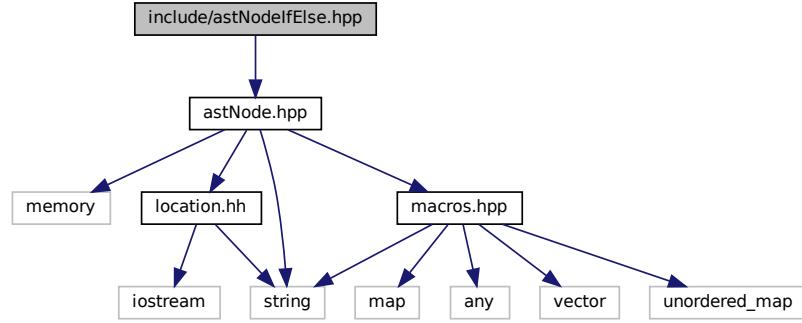
6.16.1 Detailed Description

Declare the [Tang::AstNodelentifier](#) class.

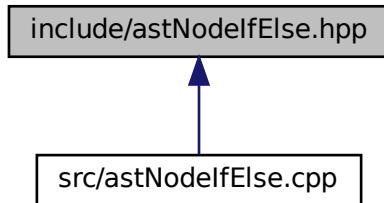
6.17 include/astNodeIfElse.hpp File Reference

Declare the [Tang::AstNodeIfElse](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeIfElse.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeIfElse](#)
An `AstNode` that represents an `if..else` statement.

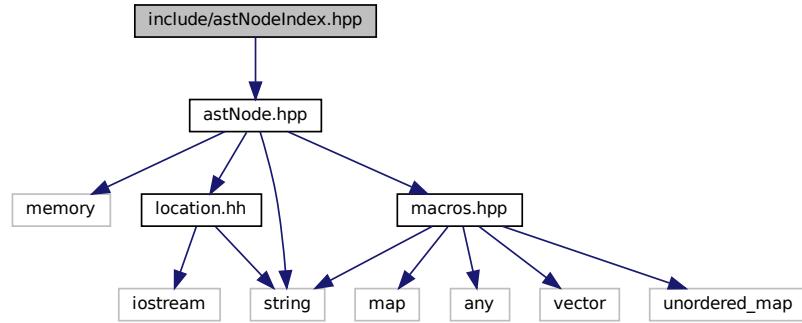
6.17.1 Detailed Description

Declare the [Tang::AstNodeIfElse](#) class.

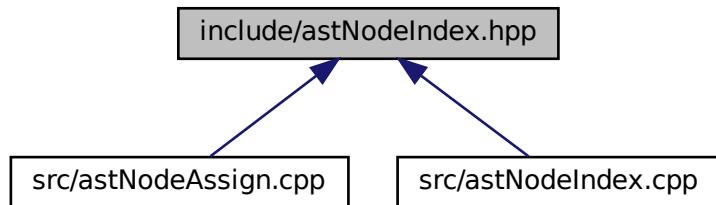
6.18 include/astNodeIndex.hpp File Reference

Declare the [Tang::AstNodeIndex](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeIndex.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeIndex](#)
An `AstNode` that represents an index into a collection.

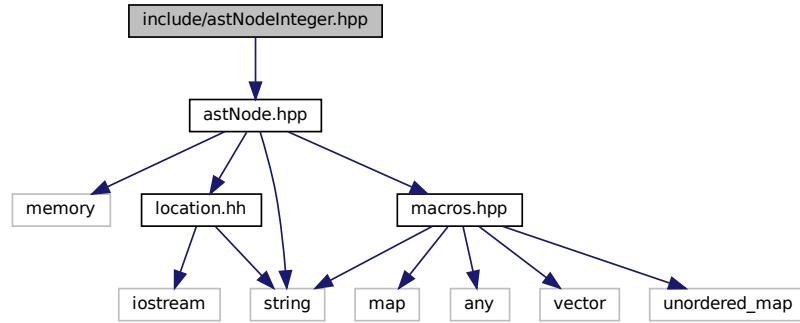
6.18.1 Detailed Description

Declare the [Tang::AstNodeIndex](#) class.

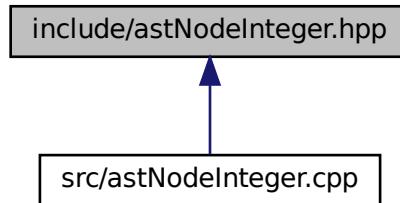
6.19 include/astNodeInteger.hpp File Reference

Declare the [Tang::AstNodeInteger](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeInteger.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeInteger](#)
An `AstNode` that represents an integer literal.

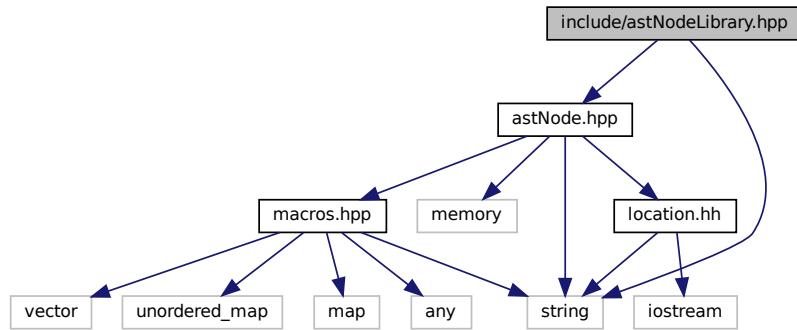
6.19.1 Detailed Description

Declare the [Tang::AstNodeInteger](#) class.

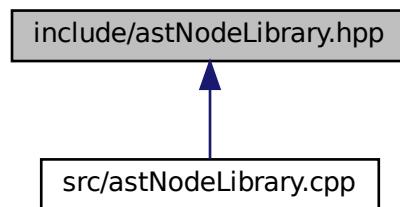
6.20 include/astNodeLibrary.hpp File Reference

Declare the [Tang::AstNodeLibrary](#) class.

```
#include <string>
#include "astNode.hpp"
Include dependency graph for astNodeLibrary.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeLibrary](#)
An `AstNode` that represents an identifier.

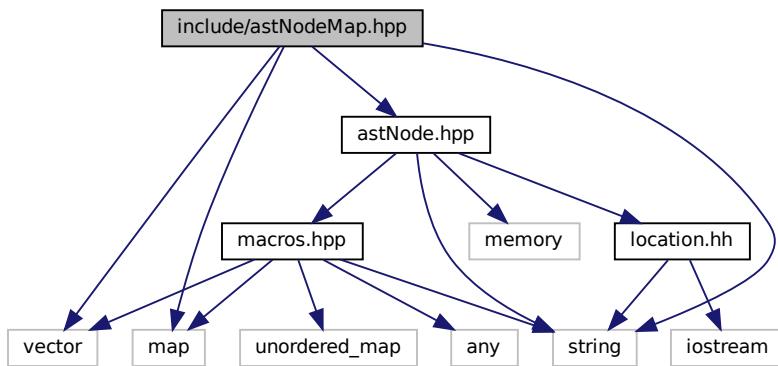
6.20.1 Detailed Description

Declare the [Tang::AstNodeLibrary](#) class.

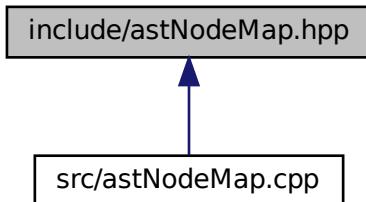
6.21 include/astNodeMap.hpp File Reference

Declare the [Tang::AstNodeMap](#) class.

```
#include <vector>
#include <map>
#include <string>
#include "astNode.hpp"
Include dependency graph for astNodeMap.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeMap](#)
An `AstNode` that represents a map literal.

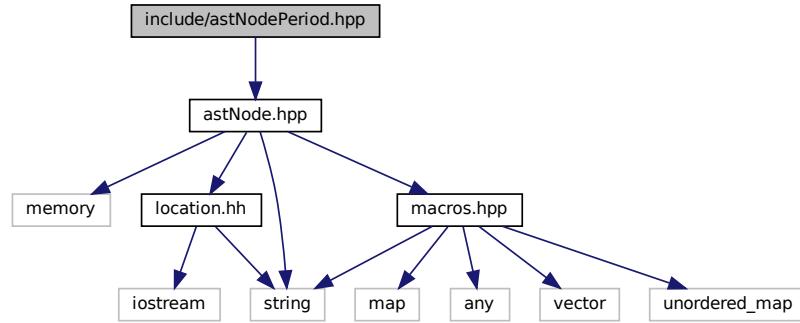
6.21.1 Detailed Description

Declare the [Tang::AstNodeMap](#) class.

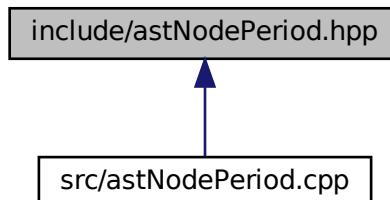
6.22 include/astNodePeriod.hpp File Reference

Declare the [Tang::AstNodePeriod](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodePeriod.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodePeriod](#)

An [AstNode](#) that represents a member access (period) into an object.

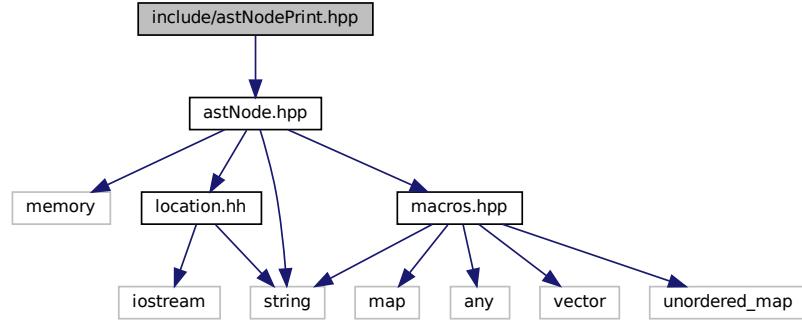
6.22.1 Detailed Description

Declare the [Tang::AstNodePeriod](#) class.

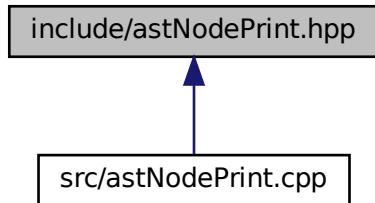
6.23 include/astNodePrint.hpp File Reference

Declare the [Tang::AstNodePrint](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodePrint.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodePrint](#)
An `AstNode` that represents a print typeeration.

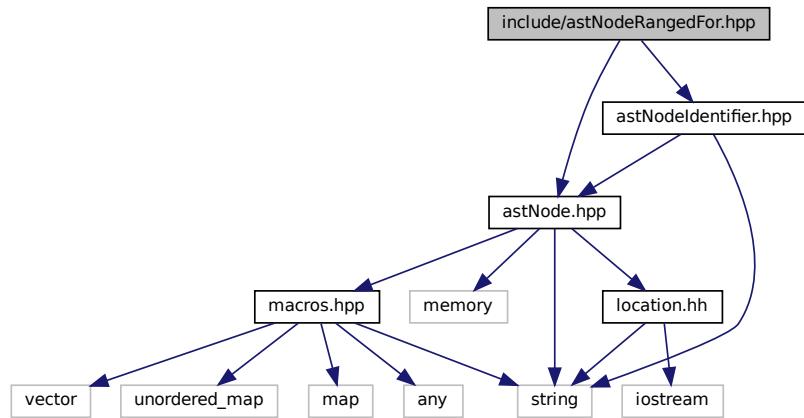
6.23.1 Detailed Description

Declare the [Tang::AstNodePrint](#) class.

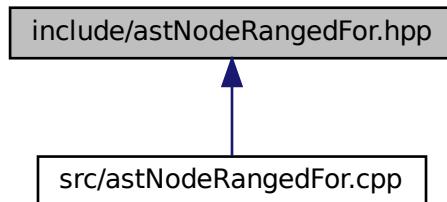
6.24 include/astNodeRangedFor.hpp File Reference

Declare the [Tang::AstNodeRangedFor](#) class.

```
#include "astNode.hpp"
#include "astNodeIdentifier.hpp"
Include dependency graph for astNodeRangedFor.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeRangedFor](#)
An `AstNode` that represents a ranged for() statement.

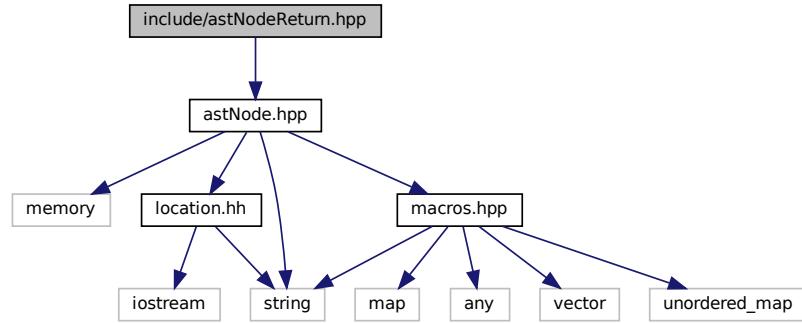
6.24.1 Detailed Description

Declare the [Tang::AstNodeRangedFor](#) class.

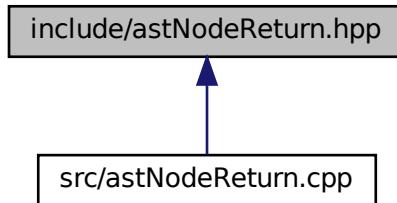
6.25 include/astNodeReturn.hpp File Reference

Declare the [Tang::AstNodeReturn](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeReturn.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeReturn](#)
An `AstNode` that represents a return statement.

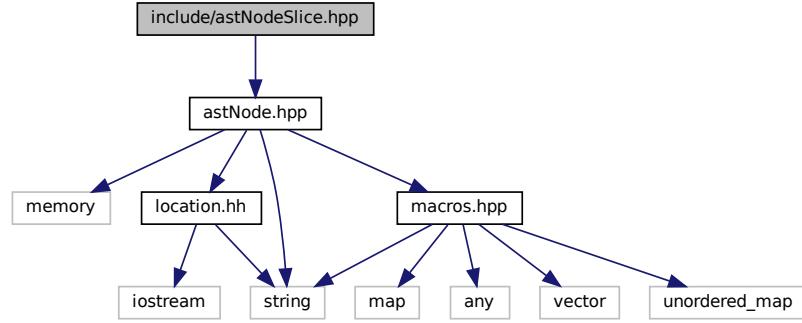
6.25.1 Detailed Description

Declare the [Tang::AstNodeReturn](#) class.

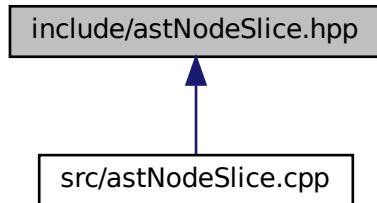
6.26 include/astNodeSlice.hpp File Reference

Declare the [Tang::AstNodeSlice](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeSlice.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeSlice](#)
An `AstNode` that represents a ternary expression.

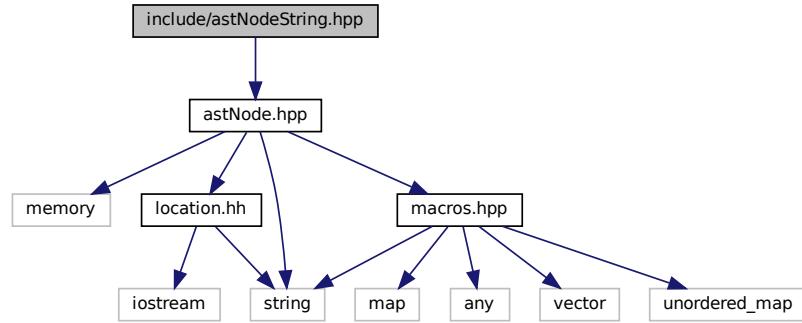
6.26.1 Detailed Description

Declare the [Tang::AstNodeSlice](#) class.

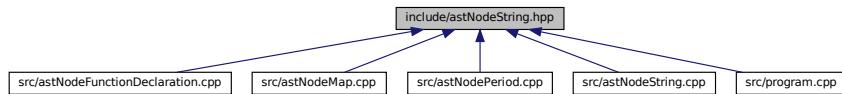
6.27 include/astNodeString.hpp File Reference

Declare the [Tang::AstNodeString](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeString.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeString](#)
An `AstNode` that represents a string literal.

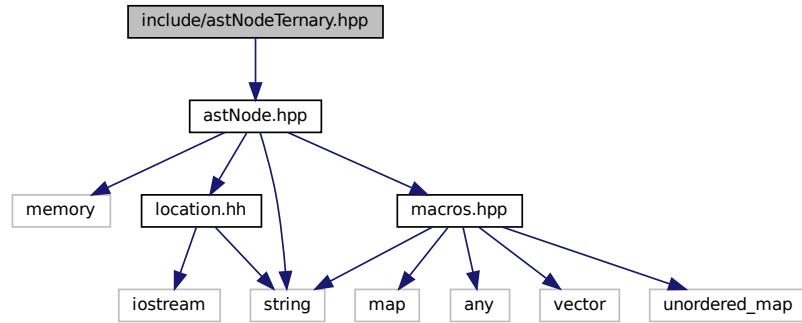
6.27.1 Detailed Description

Declare the [Tang::AstNodeString](#) class.

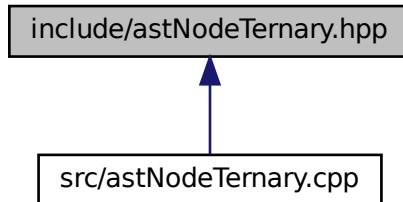
6.28 include/astNodeTernary.hpp File Reference

Declare the [Tang::AstNodeTernary](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeTernary.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeTernary](#)
An `AstNode` that represents a ternary expression.

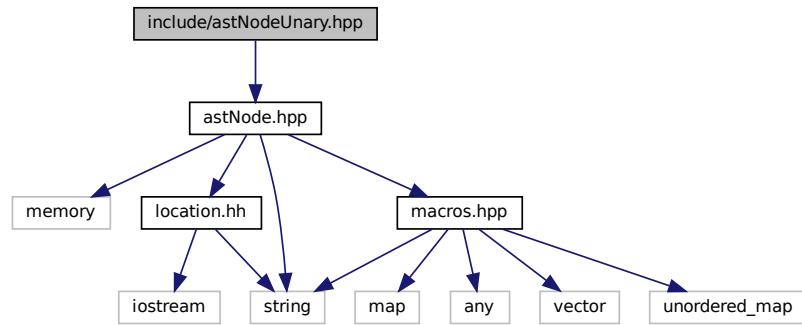
6.28.1 Detailed Description

Declare the [Tang::AstNodeTernary](#) class.

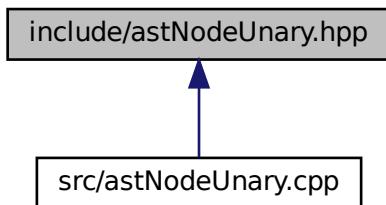
6.29 include/astNodeUnary.hpp File Reference

Declare the [Tang::AstNodeUnary](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeUnary.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeUnary](#)
An [AstNode](#) that represents a unary negation.

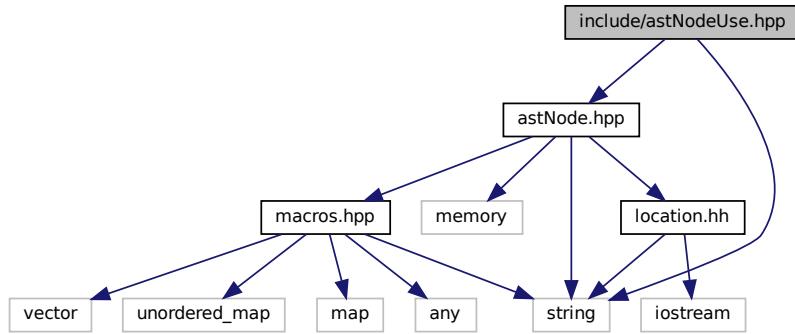
6.29.1 Detailed Description

Declare the [Tang::AstNodeUnary](#) class.

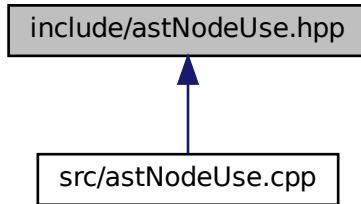
6.30 include/astNodeUse.hpp File Reference

Declare the [Tang::AstNodeUse](#) class.

```
#include <string>
#include "astNode.hpp"
Include dependency graph for astNodeUse.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeUse](#)
An `AstNode` that represents the inclusion of a library into the script.

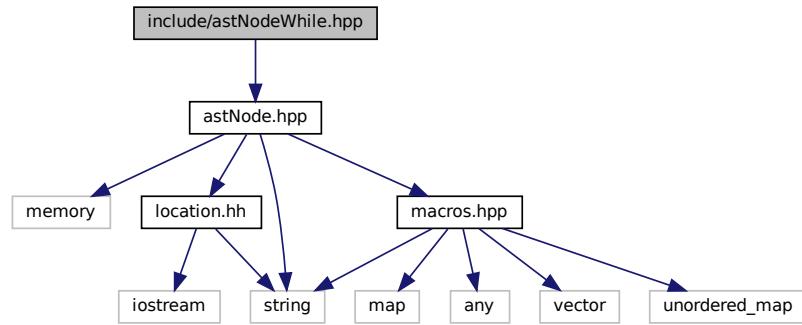
6.30.1 Detailed Description

Declare the `Tang::AstNodeUse` class.

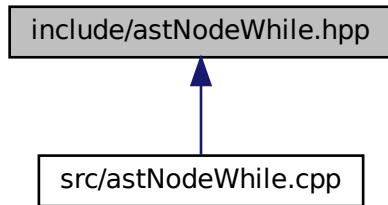
6.31 include/astNodeWhile.hpp File Reference

Declare the `Tang::AstNodeWhile` class.

```
#include "astNode.hpp"
Include dependency graph for astNodeWhile.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeWhile](#)
An `AstNode` that represents a while statement.

6.31.1 Detailed Description

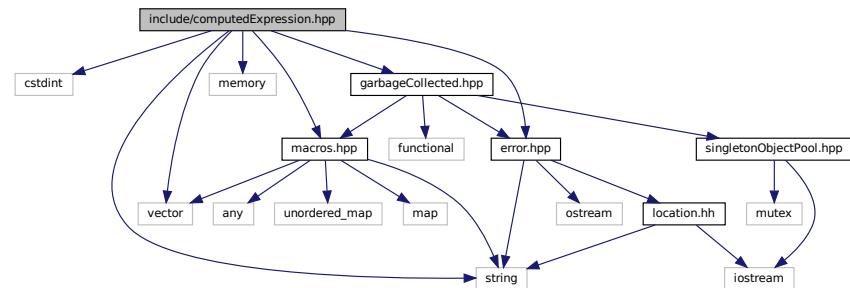
Declare the [Tang::AstNodeWhile](#) class.

6.32 include/computedExpression.hpp File Reference

Declare the [Tang::ComputedExpression](#) base class.

```
#include <cstdint>
#include <string>
```

```
#include <vector>
#include <memory>
#include "macros.hpp"
#include "garbageCollected.hpp"
#include "error.hpp"
Include dependency graph for computedExpression.hpp:
```



This graph shows which files directly or indirectly include this file:

Classes

- class [Tang::ComputedExpression](#)
Represents the result of a computation that has been executed.

6.32.1 Detailed Description

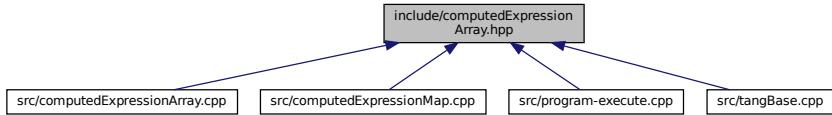
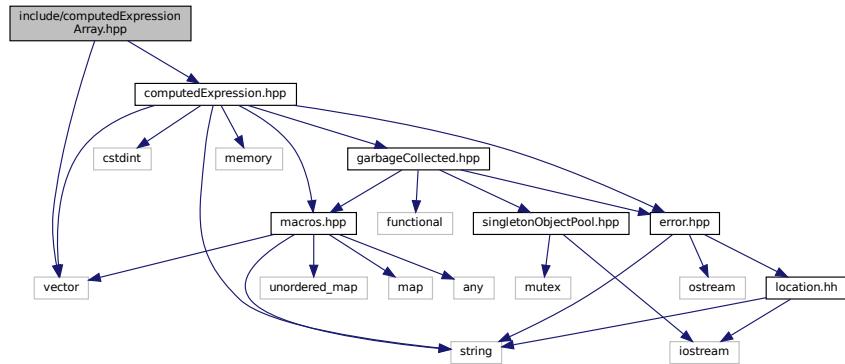
Declare the [Tang::ComputedExpression](#) base class.

6.33 include/computedExpressionArray.hpp File Reference

Declare the [Tang::ComputedExpressionArray](#) class.

```
#include <vector>
#include "computedExpression.hpp"
```

Include dependency graph for `computedExpressionArray.hpp`:



Classes

- class [Tang::ComputedExpressionArray](#)
Represents an Array that is the result of a computation.

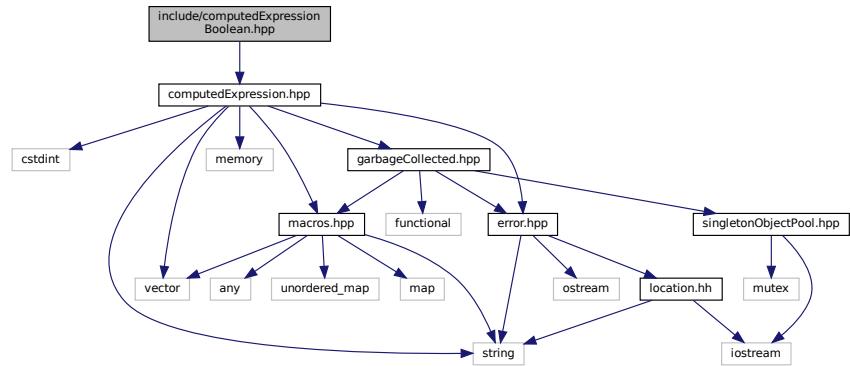
6.33.1 Detailed Description

Declare the [Tang::ComputedExpressionArray](#) class.

6.34 include/computedExpressionBoolean.hpp File Reference

Declare the [Tang::ComputedExpressionBoolean](#) class.

```
#include "computedExpression.hpp"
Include dependency graph for computedExpressionBoolean.hpp:
```



This graph shows which files directly or indirectly include this file:

Classes

- class [Tang::ComputedExpressionBoolean](#)
Represents an Boolean that is the result of a computation.

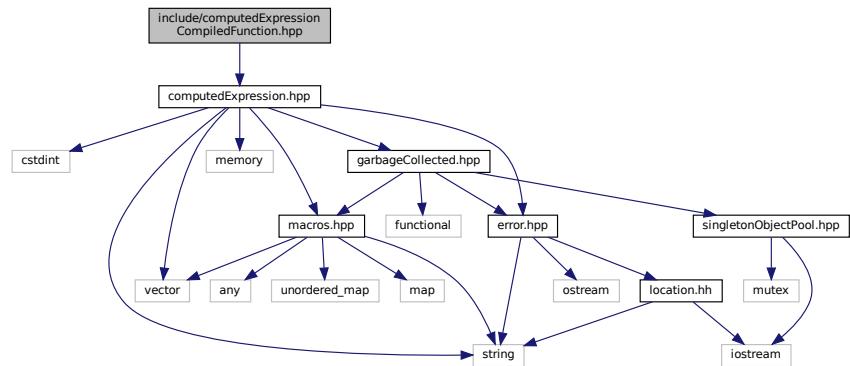
6.34.1 Detailed Description

Declare the [Tang::ComputedExpressionBoolean](#) class.

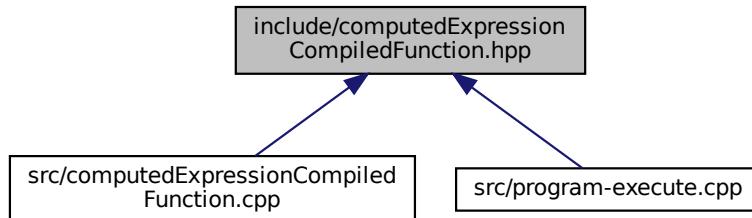
6.35 include/computedExpressionCompiledFunction.hpp File Reference

Declare the [Tang::ComputedExpressionCompiledFunction](#) class.

```
#include "computedExpression.hpp"
Include dependency graph for computedExpressionCompiledFunction.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionCompiledFunction](#)
Represents a Compiled Function declared in the script.

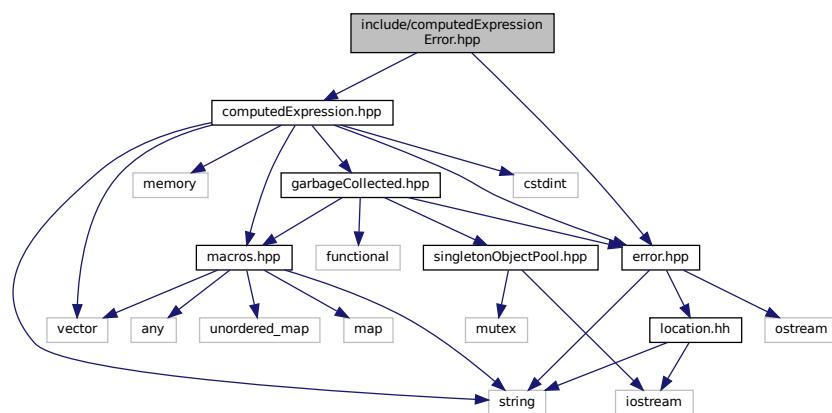
6.35.1 Detailed Description

Declare the [Tang::ComputedExpressionCompiledFunction](#) class.

6.36 include/computedExpressionError.hpp File Reference

Declare the [Tang::ComputedExpressionError](#) class.

```
#include "computedExpression.hpp"
#include "error.hpp"
Include dependency graph for computedExpressionError.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionError](#)

Represents a Runtime Error.

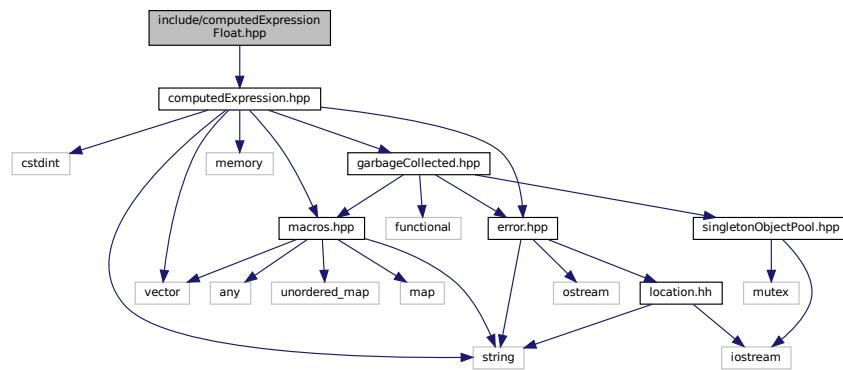
6.36.1 Detailed Description

Declare the [Tang::ComputedExpressionError](#) class.

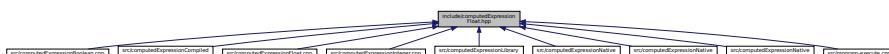
6.37 include/computedExpressionFloat.hpp File Reference

Declare the [Tang::ComputedExpressionFloat](#) class.

```
#include "computedExpression.hpp"
Include dependency graph for computedExpressionFloat.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionFloat](#)

Represents a Float that is the result of a computation.

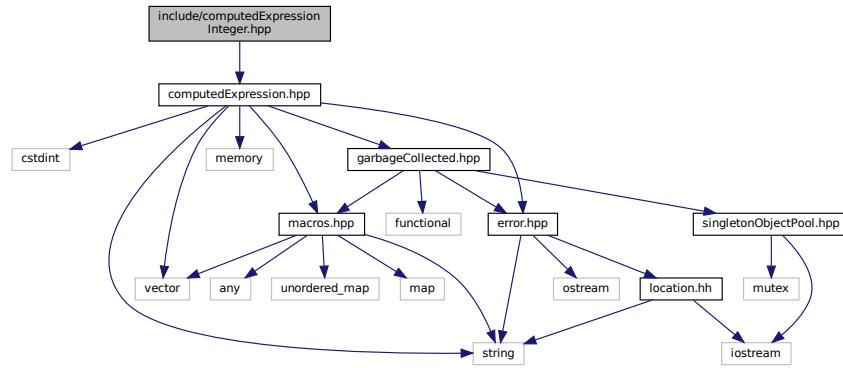
6.37.1 Detailed Description

Declare the [Tang::ComputedExpressionFloat](#) class.

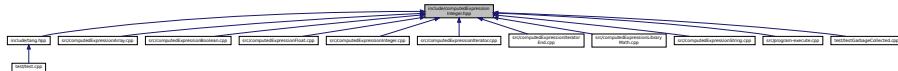
6.38 include/computedExpressionInteger.hpp File Reference

Declare the [Tang::ComputedExpressionInteger](#) class.

```
#include "computedExpression.hpp"
Include dependency graph for computedExpressionInteger.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionInteger](#)
Represents an Integer that is the result of a computation.

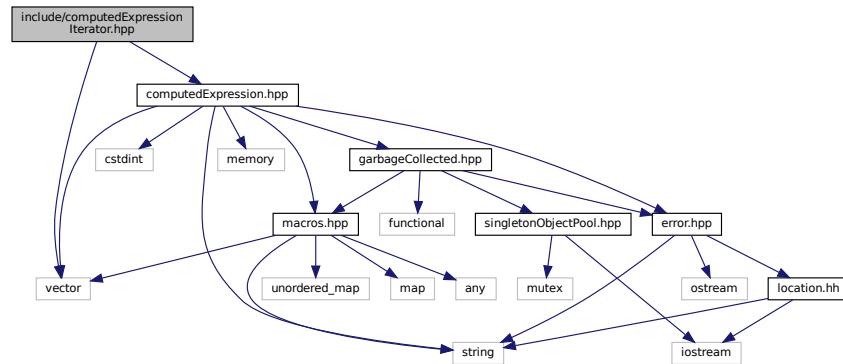
6.38.1 Detailed Description

Declare the [Tang::ComputedExpressionInteger](#) class.

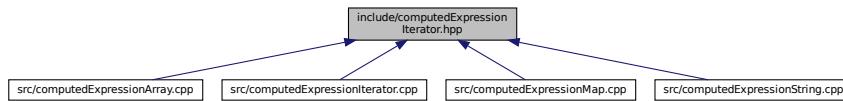
6.39 include/computedExpressionIterator.hpp File Reference

Declare the [Tang::ComputedExpressionIterator](#) class.

```
#include <vector>
#include "computedExpression.hpp"
Include dependency graph for computedExpressionIterator.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionIterator](#)
Represents an Iterator that is the result of a computation.

6.39.1 Detailed Description

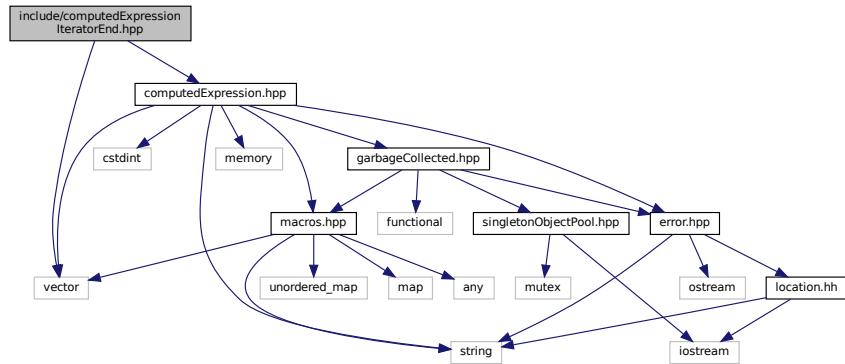
Declare the [Tang::ComputedExpressionIterator](#) class.

6.40 include/computedExpressionIteratorEnd.hpp File Reference

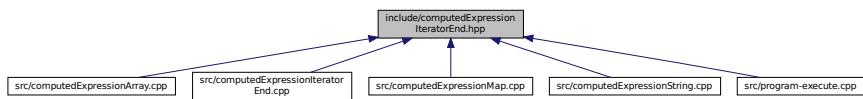
Declare the [Tang::ComputedExpressionIteratorEnd](#) class.

```
#include <vector>
#include "computedExpression.hpp"
```

Include dependency graph for `computedExpressionIteratorEnd.hpp`:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionIteratorEnd](#)
Represents that a collection has no more values through which to iterate.

6.40.1 Detailed Description

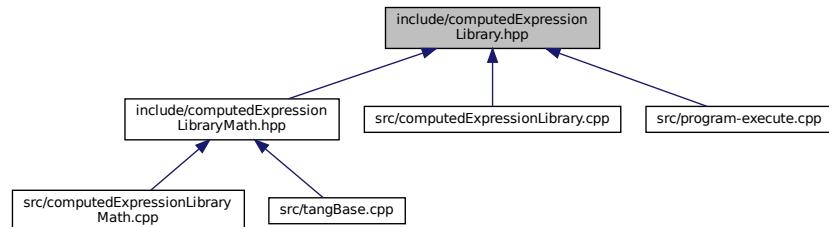
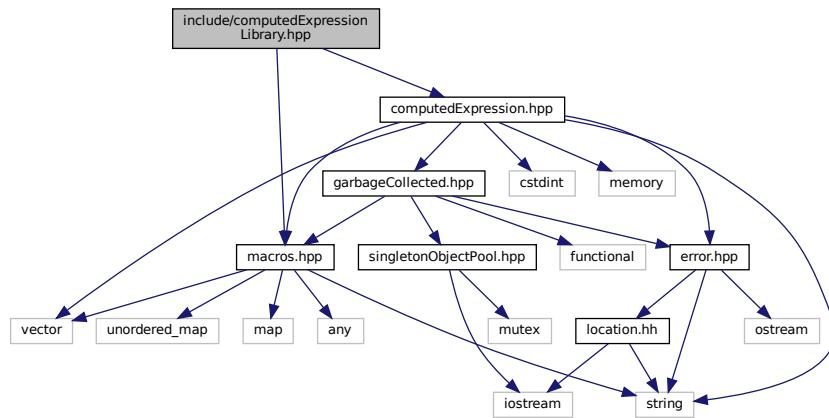
Declare the [Tang::ComputedExpressionIteratorEnd](#) class.

6.41 include/computedExpressionLibrary.hpp File Reference

Declare the [Tang::ComputedExpressionLibrary](#) class.

```
#include "macros.hpp"
#include "computedExpression.hpp"
```

Include dependency graph for computedExpressionLibrary.hpp:



Classes

- class [Tang::ComputedExpressionLibrary](#)
Represents a Runtime Library.

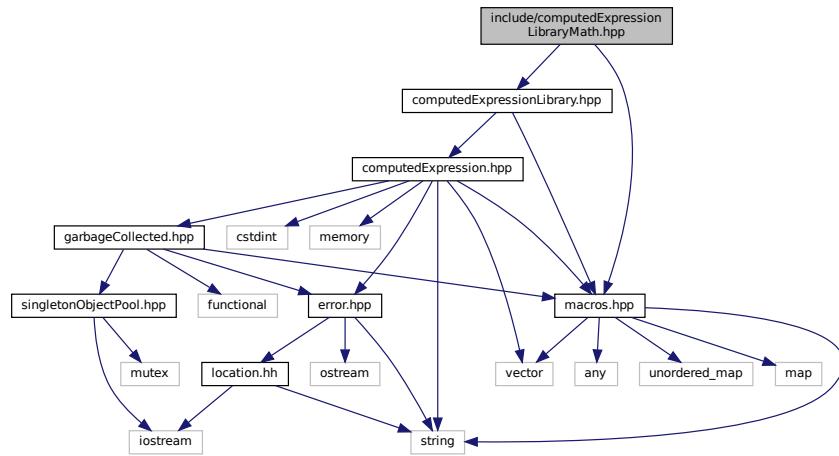
6.41.1 Detailed Description

Declare the [Tang::ComputedExpressionLibrary](#) class.

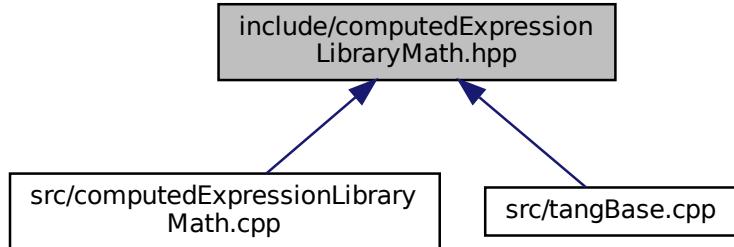
6.42 include/computedExpressionLibraryMath.hpp File Reference

Declare the [Tang::ComputedExpressionLibraryMath](#) class.

```
#include "macros.hpp"
#include "computedExpressionLibrary.hpp"
Include dependency graph for computedExpressionLibraryMath.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionLibraryMath](#)
Represents a Runtime LibraryMath.

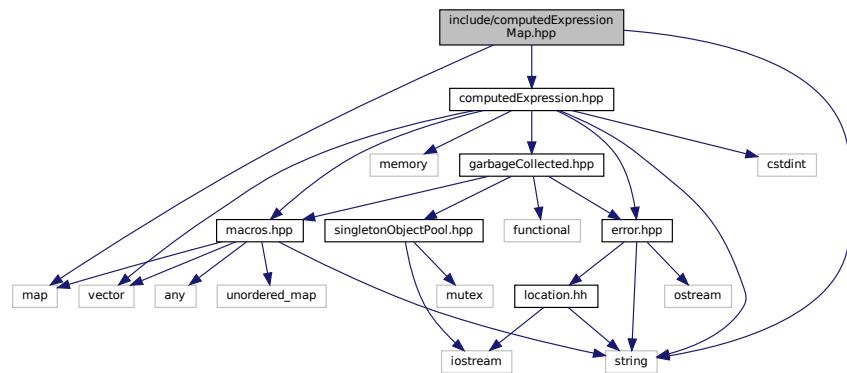
6.42.1 Detailed Description

Declare the [Tang::ComputedExpressionLibraryMath](#) class.

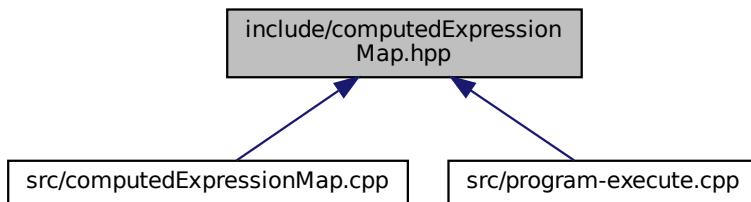
6.43 include/computedExpressionMap.hpp File Reference

Declare the [Tang::ComputedExpressionMap](#) class.

```
#include <map>
#include <string>
#include "computedExpression.hpp"
Include dependency graph for computedExpressionMap.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionMap](#)
Represents an Map that is the result of a computation.

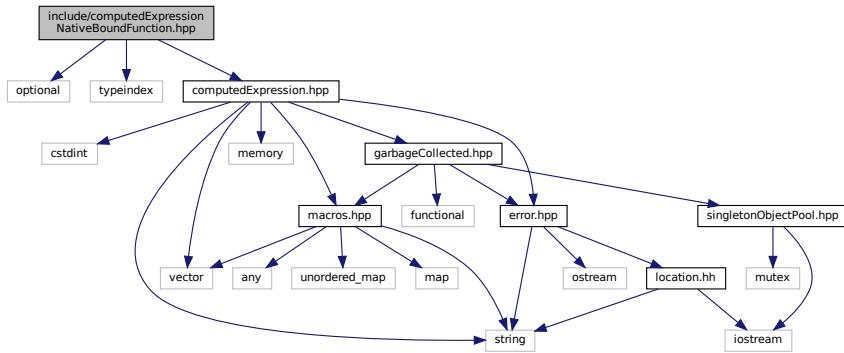
6.43.1 Detailed Description

Declare the [Tang::ComputedExpressionMap](#) class.

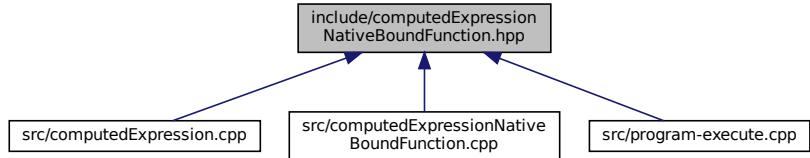
6.44 include/computedExpressionNativeBoundFunction.hpp File Reference

Declare the [Tang::ComputedExpressionNativeBoundFunction](#) class.

```
#include <optional>
#include <typeindex>
#include "computedExpression.hpp"
Include dependency graph for computedExpressionNativeBoundFunction.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionNativeBoundFunction](#)
Represents a NativeBound Function declared in the script.

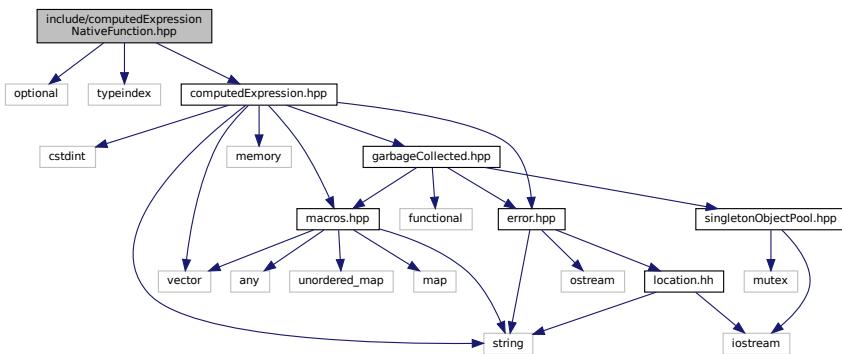
6.44.1 Detailed Description

Declare the [Tang::ComputedExpressionNativeBoundFunction](#) class.

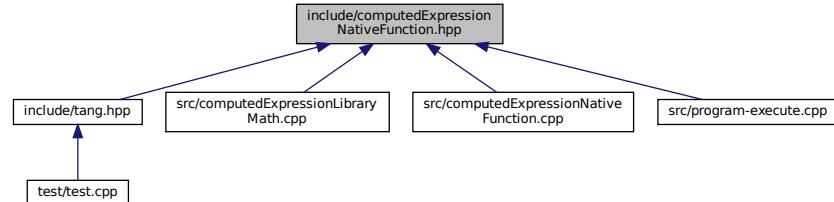
6.45 include/computedExpressionNativeFunction.hpp File Reference

Declare the [Tang::ComputedExpressionNativeFunction](#) class.

```
#include <optional>
#include <typeindex>
#include "computedExpression.hpp"
Include dependency graph for computedExpressionNativeFunction.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionNativeFunction](#)
Represents a Native Function provided by compiled C++ code.

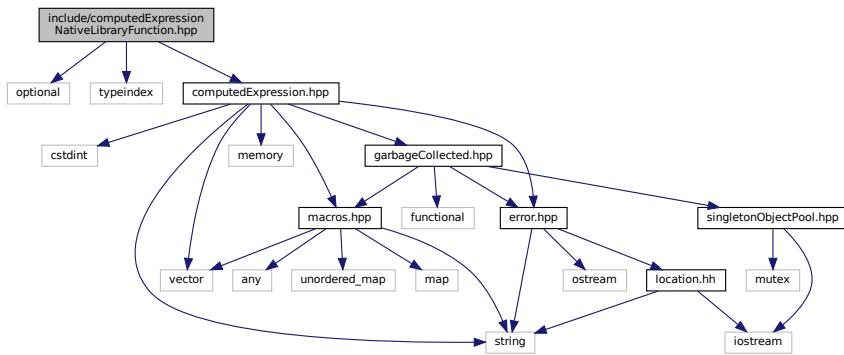
6.45.1 Detailed Description

Declare the [Tang::ComputedExpressionNativeFunction](#) class.

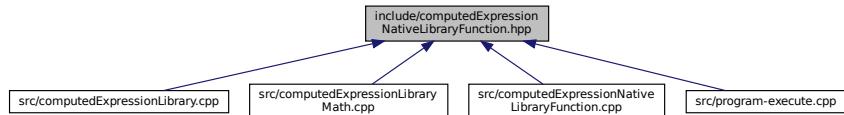
6.46 include/computedExpressionNativeLibraryFunction.hpp File Reference

Declare the [Tang::ComputedExpressionNativeLibraryFunction](#) class.

```
#include <optional>
#include <typeindex>
#include "computedExpression.hpp"
Include dependency graph for computedExpressionNativeLibraryFunction.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionNativeLibraryFunction](#)

Represents a Native Function provided by compiled C++ code that is executed to create a library or one of its attributes.

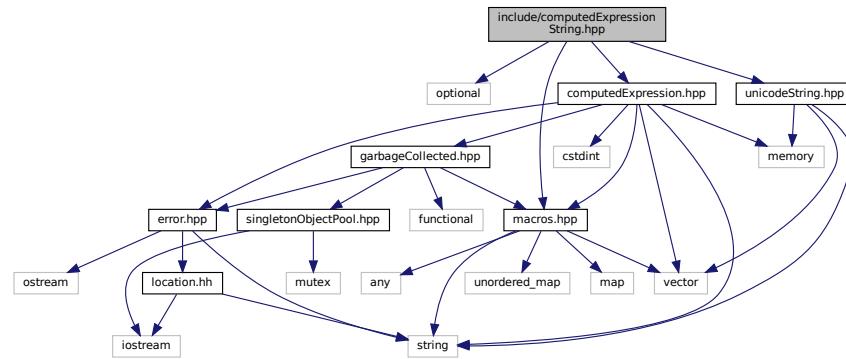
6.46.1 Detailed Description

Declare the [Tang::ComputedExpressionNativeLibraryFunction](#) class.

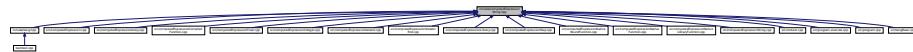
6.47 include/computedExpressionString.hpp File Reference

Declare the `Tang::ComputedExpressionString` class.

```
#include <optional>
#include "macros.hpp"
#include "computedExpression.hpp"
#include "unicodeString.hpp"
Include dependency graph for computedExpressionString.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionString](#)
Represents a String that is the result of a computation.

6.47.1 Detailed Description

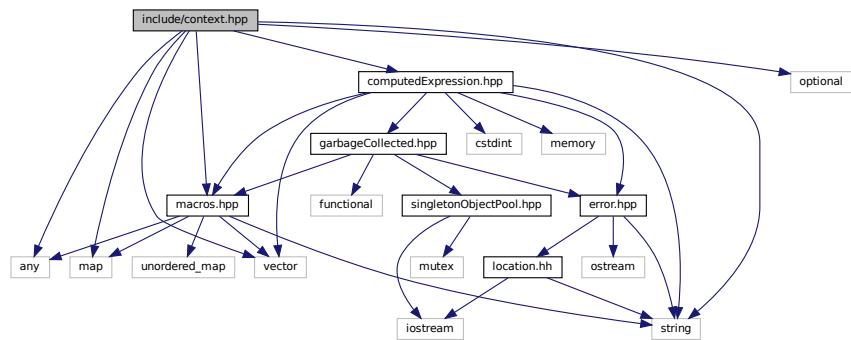
Declare the `Tang::ComputedExpressionString` class.

6.48 include/context.hpp File Reference

Declare the `Tang::Context` class.

```
#include <any>
#include <map>
#include <optional>
#include <string>
#include <vector>
```

```
#include "macros.hpp"
#include "computedExpression.hpp"
Include dependency graph for context.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::Context](#)
Holds all environment variables specific to the execution of a program.

6.48.1 Detailed Description

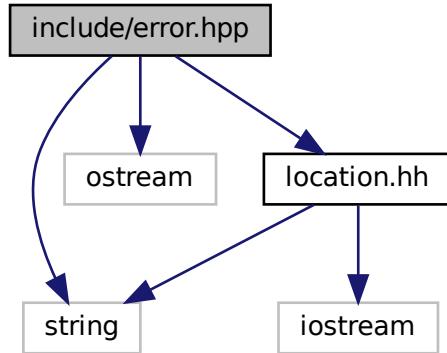
Declare the [Tang::Context](#) class.

6.49 include/error.hpp File Reference

Declare the [Tang::Error](#) class used to describe syntax and runtime errors.

```
#include <string>
#include <ostream>
```

```
#include "location.hpp"
Include dependency graph for error.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::Error](#)

The [Error](#) class is used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error.

6.49.1 Detailed Description

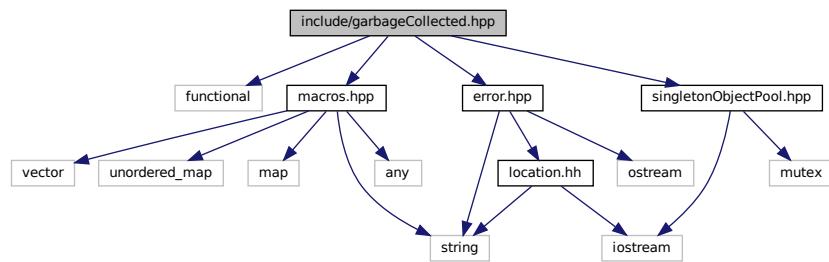
Declare the [Tang::Error](#) class used to describe syntax and runtime errors.

6.50 include/garbageCollected.hpp File Reference

Declare the [Tang::GarbageCollected](#) class.

```
#include <functional>
#include "macros.hpp"
#include "singletonObjectPool.hpp"
```

```
#include "error.hpp"
Include dependency graph for garbageCollected.hpp:
```



This graph shows which files directly or indirectly include this file:

Classes

- class [Tang::GarbageCollected](#)
A container that acts as a resource-counting garbage collector for the specified type.

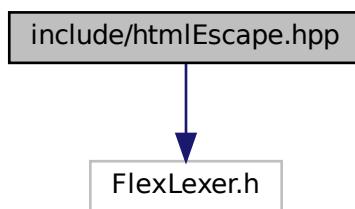
6.50.1 Detailed Description

Declare the [Tang::GarbageCollected](#) class.

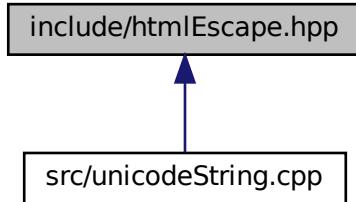
6.51 include/htmlEscape.hpp File Reference

Declare the [Tang::HtmlEscape](#) used to tokenize a Tang script.

```
#include <FlexLexer.h>
Include dependency graph for htmlEscape.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::HtmlEscape](#)
The Flex lexer class for the main Tang language.

Macros

- `#define yyFlexLexer TangHtmlEscapeFlexLexer`
- `#define YY_DECL std::string Tang::HtmlEscape::get_next_token()`

6.51.1 Detailed Description

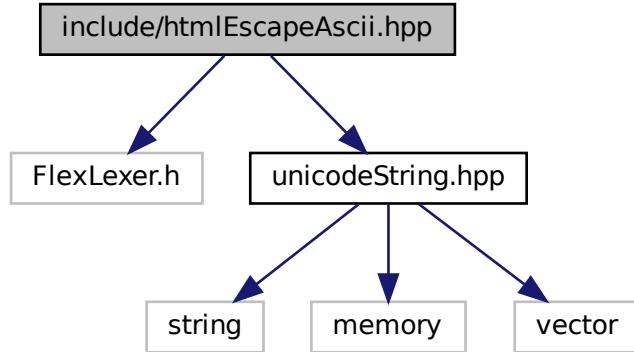
Declare the [Tang::HtmlEscape](#) used to tokenize a Tang script.

6.52 include/htmlEscapeAscii.hpp File Reference

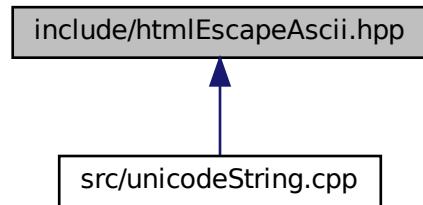
Declare the [Tang::HtmlEscapeAscii](#) used to tokenize a Tang script.

```
#include <FlexLexer.h>
#include "unicodeString.hpp"
```

Include dependency graph for htmlEscapeAscii.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::HtmlEscapeAscii](#)
The Flex lexer class for the main Tang language.

Macros

- `#define yyFlexLexer TangHtmlEscapeAsciiFlexLexer`
- `#define YY_DECL std::string Tang::HtmlEscapeAscii::get_next_token()`

6.52.1 Detailed Description

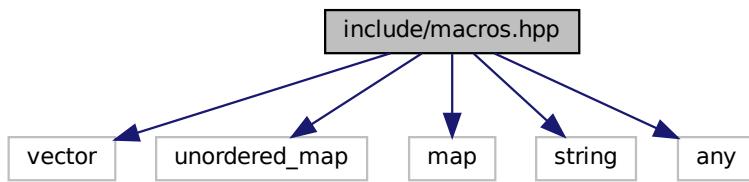
Declare the [Tang::HtmlEscapeAscii](#) used to tokenize a Tang script.

6.53 include/macros.hpp File Reference

Contains generic macros.

```
#include <vector>
#include <unordered_map>
#include <map>
#include <string>
#include <any>
```

Include dependency graph for macros.hpp:



This graph shows which files directly or indirectly include this file:



Typedefs

- using `Tang::integer_t` = `int32_t`
Define the size of signed integers used by Tang.
- using `Tang::uinteger_t` = `int32_t`
Define the size of integers used by Tang.
- using `Tang::float_t` = `float`
Define the size of floats used by Tang.
- using `Tang::LibraryFunction` = `GarbageCollected(*)(Context &)`
A function pointer that will be executed.
- using `Tang::NativeFunction` = `GarbageCollected(*)(std::vector< GarbageCollected > &, Context &)`
A function pointer that will be executed as bound to an object.
- using `Tang::NativeBoundFunction` = `GarbageCollected(*)(GarbageCollected &, std::vector< GarbageCollected > &)`
A function pointer that will be executed as bound to an object.
- using `Tang::LibraryFunctionMap` = `std::map< std::string, LibraryFunction >`
A map of method names to LibraryFunction objects.
- using `Tang::NativeBoundFunctionMap` = `std::map< std::string, std::pair< size_t, NativeBoundFunction > >`
A map of method names to NativeBoundFunction objects.
- using `Tang::ContextData` = `std::unordered_map< std::string, std::any >`
Used to hold arbitrary data which should be made available to a program during the program execution.

6.53.1 Detailed Description

Contains generic macros.

6.54 include/opcode.hpp File Reference

Declare the Opcodes used in the Bytecode representation of a program.

This graph shows which files directly or indirectly include this file:



Enumerations

- enum class `Tang::Opcode` {
 `POP` , `PEEK` , `POKE` , `COPY` ,
 `JMP` , `JMPF` , `JMPF_POP` , `JMPT` ,
 `JMPT_POP` , `NULLVAL` , `INTEGER` , `FLOAT` ,
 `BOOLEAN` , `STRING` , `ARRAY` , `MAP` ,
 `LIBRARY` , `LIBRARYSAVE` , `LIBRARYCOPY` , `FUNCTION` ,
 `ASSIGNINDEX` , `ADD` , `SUBTRACT` , `MULTIPLY` ,
 `DIVIDE` , `MODULO` , `NEGATIVE` , `NOT` ,
 `LT` , `LTE` , `GT` , `GTE` ,
 `EQ` , `NEQ` , `PERIOD` , `INDEX` ,
 `SLICE` , `GETITERATOR` , `ITERATORNEXT` , `ISITERATOREND` ,
 `CASTINTEGER` , `CASTFLOAT` , `CASTBOOLEAN` , `CASTSTRING` ,
 `CALLFUNC` , `RETURN` , `PRINT` }

6.54.1 Detailed Description

Declare the Opcodes used in the Bytecode representation of a program.

6.54.2 Enumeration Type Documentation

6.54.2.1 Opcode

```
enum Tang::Opcode [strong]
```

Enumerator

<code>POP</code>	Pop a val.
<code>PEEK</code>	Stack # (from fp): push val from stack #.
<code>POKE</code>	Stack # (from fp): Copy a val, store @ stack #.
<code>COPY</code>	Stack # (from fp): Deep copy val @ stack #, store @ stack #.

Enumerator

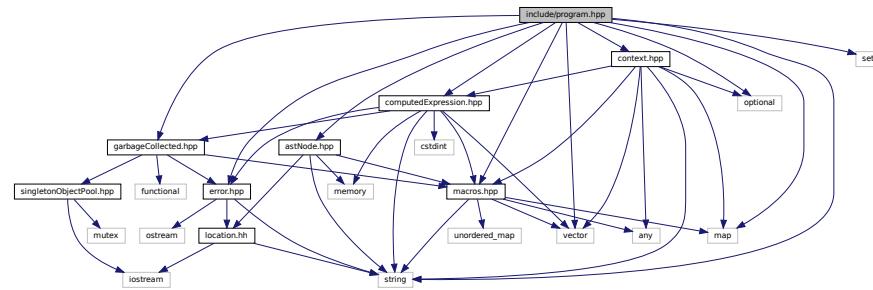
JMP	PC #: set pc to PC #.
JMPF	PC #: read val, if false, set pc to PC #.
JMPF_POP	PC #: pop val, if false, set pc to PC #.
JMPT	PC #: read val, if true, set pc to PC #.
JMPT_POP	PC #: pop val, if true, set pc to PC #.
NULLVAL	Push a null onto the stack.
INTEGER	Push an integer onto the stack.
FLOAT	Push a floating point number onto the stack.
BOOLEAN	Push a boolean onto the stack.
STRING	Get len, char string: push string.
ARRAY	Get len, pop len items, putting them into an array with the last array item popped first.
MAP	Get len, pop len value then key pairs, putting them into a map.
LIBRARY	Pop name, push Library identified by name.
LIBRARYSAVE	Get index, save top of stack to library[index].
LIBRARYCOPY	Get index, load from library[index].
FUNCTION	Get argc, PC#: push function(argc, PC #)
ASSIGNINDEX	Pop index, pop collection, pop value, push (collection[index] = value)
ADD	Pop rhs, pop lhs, push lhs + rhs.
SUBTRACT	Pop rhs, pop lhs, push lhs - rhs.
MULTIPLY	Pop rhs, pop lhs, push lhs * rhs.
DIVIDE	Pop rhs, pop lhs, push lhs / rhs.
MODULO	Pop rhs, pop lhs, push lhs % rhs.
NEGATIVE	Pop val, push negative val.
NOT	Pop val, push logical not of val.
LT	Pop rhs, pop lhs, push lhs < rhs.
LTE	Pop rhs, pop lhs, push lhs <= rhs.
GT	Pop rhs, pop lhs, push lhs > rhs.
GTE	Pop rhs, pop lhs, push lhs >= rhs.
EQ	Pop rhs, pop lhs, push lhs == rhs.
NEQ	Pop rhs, pop lhs, push lhs != rhs.
PERIOD	Pop rhs, pop lhs, push lhs.rhs.
INDEX	Pop index, pop collection, push collection[index].
SLICE	Pop skip, pop end, pop begin, pop collection, push collection[begin:end:skip].
GETITERATOR	Pop a collection, push the collection iterator.
ITERATORNEXT	Pop an iterator, push the next iterator value.
ISITERATOREND	Pop a val, push bool(is val == iterator end)
CASTINTEGER	Pop a val, typecast to int, push.
CASTFLOAT	Pop a val, typecast to float, push.
CASTBOOLEAN	Pop a val, typecast to boolean, push.
CASTSTRING	Pop a val, typecast to string, push.
CALLFUNC	Get argc, Pop a function, execute function if argc matches.
RETURN	Get stack #, pop return val, pop (stack #) times, push val, restore fp, restore pc.
PRINT	Pop val, print(val), push error or NULL.

6.55 include/program.hpp File Reference

Declare the [Tang::Program](#) class used to compile and execute source code.

```
#include <string>
#include <optional>
#include <vector>
#include <set>
#include <map>
#include "macros.hpp"
#include "astNode.hpp"
#include "error.hpp"
#include "computedExpression.hpp"
#include "garbageCollected.hpp"
#include "context.hpp"
Include dependency graph for program.hpp:
```

Include dependency graph for program.hpp:



This graph shows which files directly or indirectly include this file:

Classes

- class `Tang::Program`
Represents a compiled script or template that may be executed.

Typedefs

- using `Tang::Bytecode` = `std::vector< Tang::uinteger_t >`
Contains the Opcodes of a compiled program.

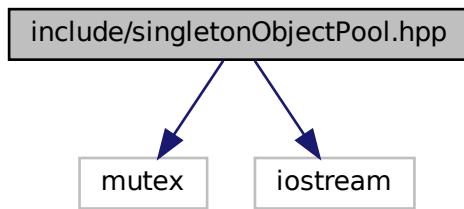
6.55.1 Detailed Description

Declare the `Tang::Program` class used to compile and execute source code.

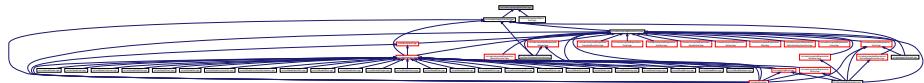
6.56 include/singletonObjectPool.hpp File Reference

Declare the [Tang::SingletonObjectPool](#) class.

```
#include <mutex>
#include <iostream>
Include dependency graph for singletonObjectPool.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::SingletonObjectPool< T >](#)
A thread-safe, singleton object pool of the designated type.

Macros

- `#define GROW 1024`
The threshold size to use when allocating blocks of data, measured in the number of instances of the object type.

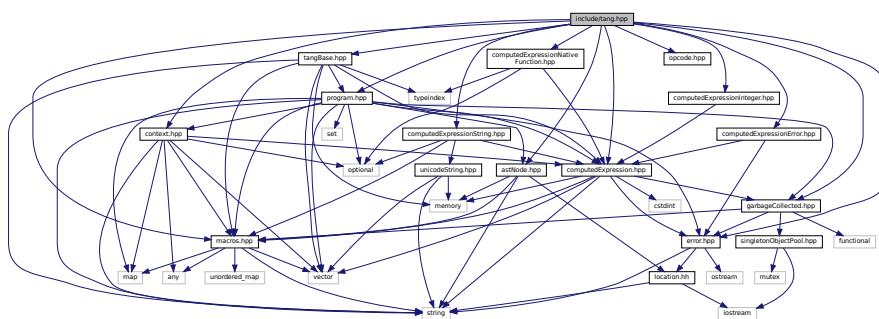
6.56.1 Detailed Description

Declare the [Tang::SingletonObjectPool](#) class.

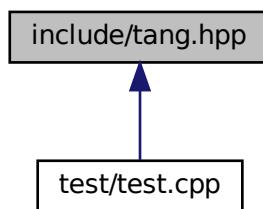
6.57 include/tang.hpp File Reference

Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

```
#include "macros.hpp"
#include "tangBase.hpp"
#include "astNode.hpp"
#include "error.hpp"
#include "garbageCollected.hpp"
#include "program.hpp"
#include "context.hpp"
#include "opcode.hpp"
#include "computedExpression.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionNativeFunction.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionError.hpp"
Include dependency graph for tang.hpp:
```



This graph shows which files directly or indirectly include this file:



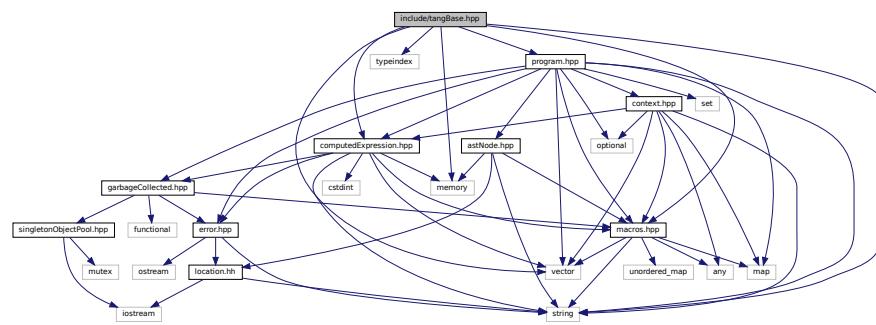
6.57.1 Detailed Description

Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

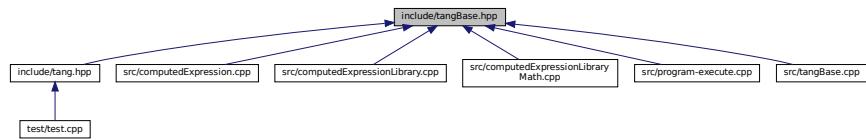
6.58 include/tangBase.hpp File Reference

Declare the [Tang::TangBase](#) class used to interact with Tang.

```
#include <memory>
#include <string>
#include <typeindex>
#include <vector>
#include "macros.hpp"
#include "program.hpp"
#include "computedExpression.hpp"
Include dependency graph for tangBase.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::TangBase](#)
The base class for the Tang programming language.

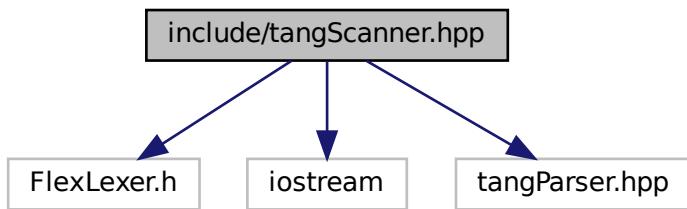
6.58.1 Detailed Description

Declare the [Tang::TangBase](#) class used to interact with Tang.

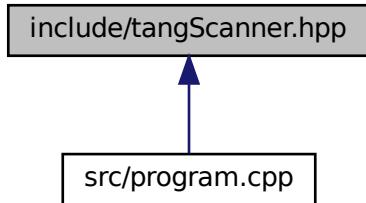
6.59 include/tangScanner.hpp File Reference

Declare the [Tang::TangScanner](#) used to tokenize a Tang script.

```
#include <FlexLexer.h>
#include <iostream>
#include "tangParser.hpp"
Include dependency graph for tangScanner.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::TangScanner](#)
The Flex lexer class for the main Tang language.

Macros

- `#define yyFlexLexer TangTangFlexLexer`
- `#define YY_DECL Tang::TangParser::symbol_type Tang::TangScanner::get_next_token()`

6.59.1 Detailed Description

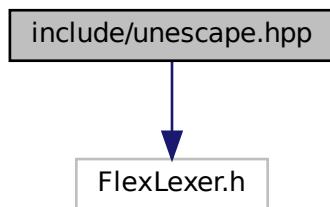
Declare the [Tang::TangScanner](#) used to tokenize a Tang script.

6.60 include/unescape.hpp File Reference

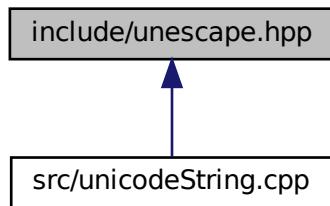
Declare the [Tang::Unescape](#) used to tokenize a Tang script.

```
#include <FlexLexer.h>
```

Include dependency graph for unescape.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::Unescape](#)
The Flex lexer class for the main Tang language.

Macros

- `#define yyFlexLexer TangUnescapeFlexLexer`
- `#define YY_DECL std::string Tang::Unescape::get_next_token()`

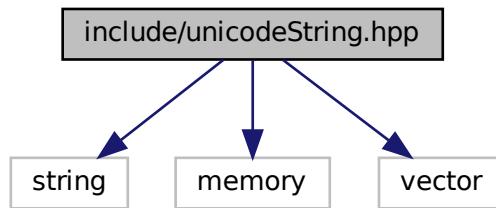
6.60.1 Detailed Description

Declare the [Tang::Unescape](#) used to tokenize a Tang script.

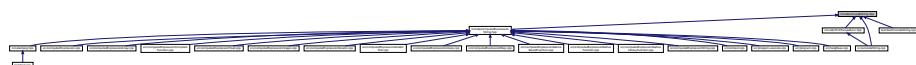
6.61 include/unicodeString.hpp File Reference

Contains the code to interface with the ICU library.

```
#include <string>
#include <memory>
#include <vector>
Include dependency graph for unicodeString.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::UnicodeString](#)
Represents a UTF-8 encoded string that is Unicode-aware.

Functions

- std::string [Tang::unescape](#) (const std::string &str)
Return an "unescaped" version of the provided string, which, when interpreted by Tang, should result in a representation equivalent to the original source string.
- std::string [Tang::htmlEscape](#) (const std::string &str)
Return an "html escaped" version of the provided string.
- std::string [Tang::htmlEscapeAscii](#) (const std::string &str, [UnicodeString::Type](#) type=[UnicodeString::Type::Untrusted](#))
Return an Ascii-only, "html escaped" version of the provided string.

6.61.1 Detailed Description

Contains the code to interface with the ICU library.

6.61.2 Function Documentation

6.61.2.1 htmlEscape()

```
string Tang::htmlEscape (
    const std::string & str )
```

Return an "html escaped" version of the provided string.

Only "critical" characters <, >, &, ", and `` will be escaped. All other characters will be allowed through unaltered. The result is a UTF-8 encoded string that is safe for inclusion in an HTML template without disturbing the HTML structure.

Parameters

<code>str</code>	The string to be escaped.
------------------	---------------------------

Returns

An "escaped" version of the provided string.

Here is the call graph for this function:



6.61.2.2 htmlEscapeAscii()

```
string Tang::htmlEscapeAscii (
    const std::string & str,
    UnicodeString::Type type = UnicodeString::Type::Untrusted )
```

Return an Ascii-only, "html escaped" version of the provided string.

This function will convert all characters into an Ascii-only representation of the provided UTF-8 encoded string. Visible, standard Ascii characters will pass through unaltered, but all others will be replaced by their HTML escape sequence (if it exists), or the appropriate hexadecimal escape code.

Parameters

<code>str</code>	The string to be escaped.
------------------	---------------------------

Returns

An "escaped" version of the provided string.

Here is the call graph for this function:



6.61.2.3 `unescape()`

```
string Tang::unescape (
    const std::string & str )
```

Return an "unescaped" version of the provided string, which, when interpreted by Tang, should result in a representation equivalent to the original source string.

Parameters

<code>str</code>	The string to be unescaped.
------------------	-----------------------------

Returns

An "unescaped" version of the provided string.

Here is the call graph for this function:

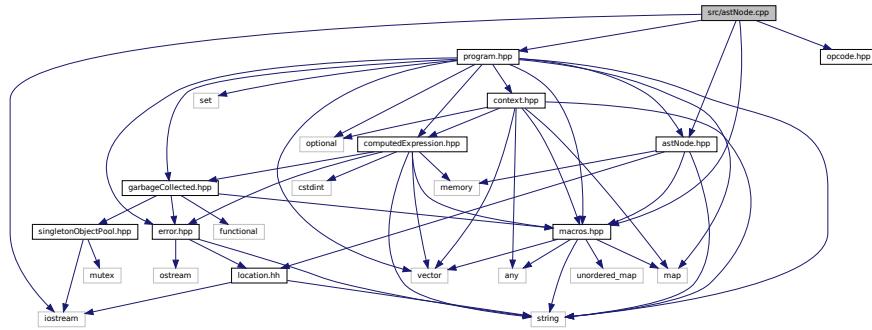


6.62 src/astNode.cpp File Reference

Define the `Tang::AstNode` class.

```
#include <iostream>
#include "macros.hpp"
#include "astNode.hpp"
#include "program.hpp"
#include "opcode.hpp"
Include dependency graph for astNode.cpp:
```

Include dependency graph for astNode.cpp:



6.62.1 Detailed Description

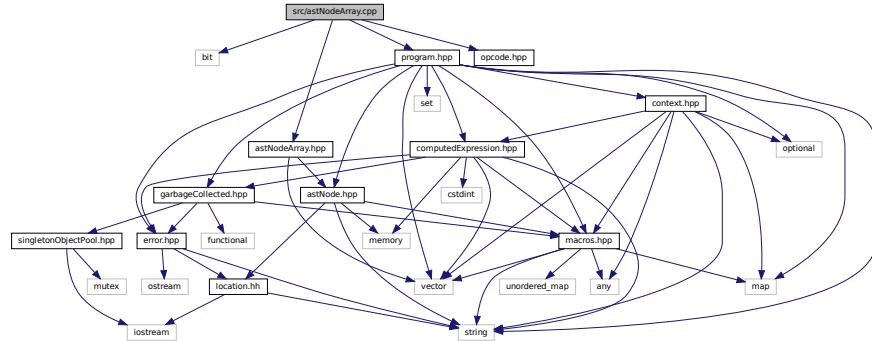
Define the `Tang::AstNode` class.

6.63 src/astNodeArray.cpp File Reference

Define the `Tang::AstNodeArray` class.

```
#include <bit>
#include "astNodeArray.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeArray.hpp:
```

Include dependency graph for astNodeArray.cpp:



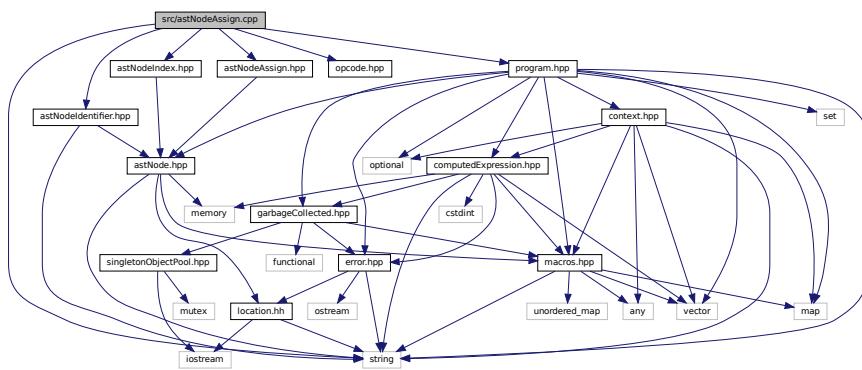
6.63.1 Detailed Description

Define the [Tang::AstNodeArray](#) class.

6.64 src/astNodeAssign.cpp File Reference

Define the [Tang::AstNodeAssign](#) class.

```
#include <string>
#include "astNodeAssign.hpp"
#include "astNodeIdentifier.hpp"
#include "astNodeIndex.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeAssign.cpp:
```



6.64.1 Detailed Description

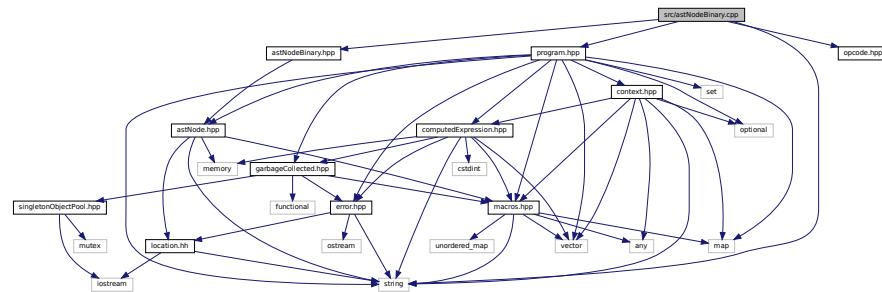
Define the [Tang::AstNodeAssign](#) class.

6.65 src/astNodeBinary.cpp File Reference

Define the [Tang::AstNodeBinary](#) class.

```
#include <string>
#include "astNodeBinary.hpp"
#include "opcode.hpp"
```

```
#include "program.hpp"
Include dependency graph for astNodeBinary.cpp:
```



6.65.1 Detailed Description

Define the `Tang::AstNodeBinary` class.

6.66 src/astNodeBlock.cpp File Reference

Define the `Tang::AstNodeBlock` class.

```
#include <bit>
#include "astNodeBlock.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeBlock.cpp:
```

6.66.1 Detailed Description

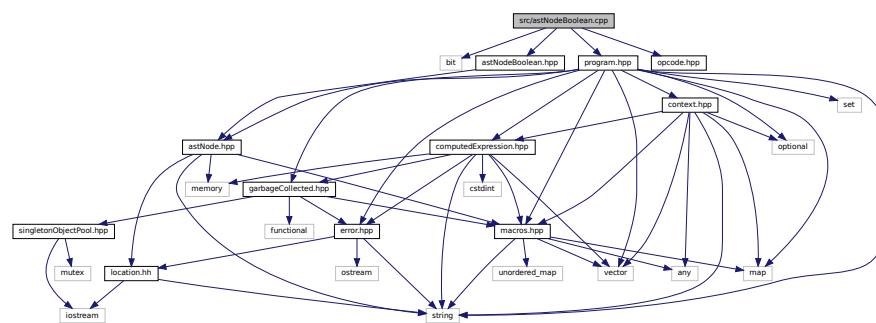
Define the `Tang::AstNodeBlock` class.

6.67 src/astNodeBoolean.cpp File Reference

Define the [Tang::AstNodeBoolean](#) class.

```
#include <bit>
#include "astNodeBoolean.hpp"
#include "opcode.hpp"
#include "program.hpp"
```

Include dependency graph for astNodeBoolean.cpp:



6.67.1 Detailed Description

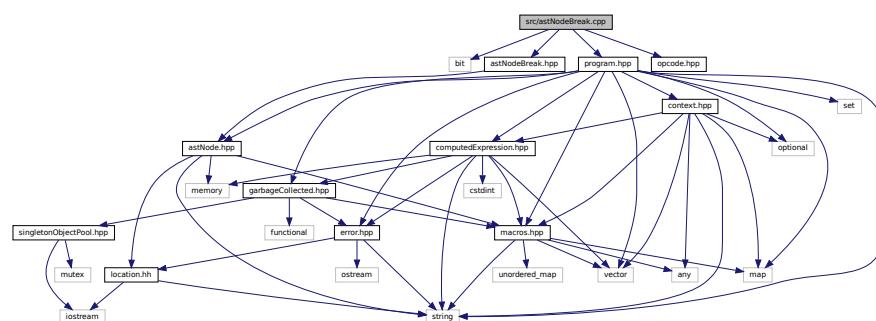
Define the [Tang::AstNodeBoolean](#) class.

6.68 src/astNodeBreak.cpp File Reference

Define the [Tang::AstNodeBreak](#) class.

```
#include <bit>
#include "astNodeBreak.hpp"
#include "opcode.hpp"
#include "program.hpp"
```

Include dependency graph for astNodeBreak.cpp:



6.68.1 Detailed Description

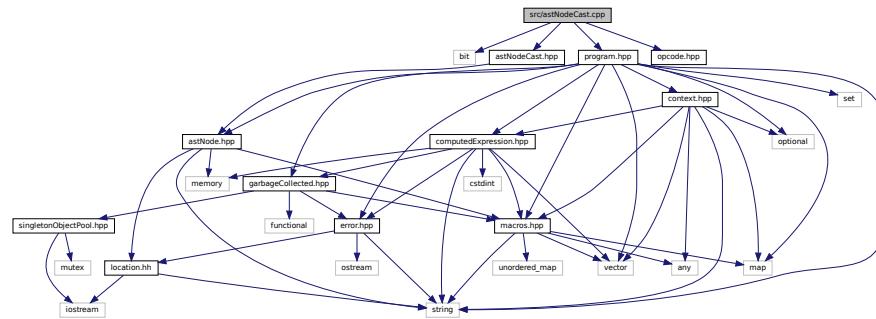
Define the `Tang::AstNodeBreak` class.

6.69 src/astNodeCast.cpp File Reference

Define the `Tang::AstNodeCast` class.

```
#include <bit>
#include "astNodeCast.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeCast.cpp:
```

Include dependency graph for astNodeCast.cpp:



6.69.1 Detailed Description

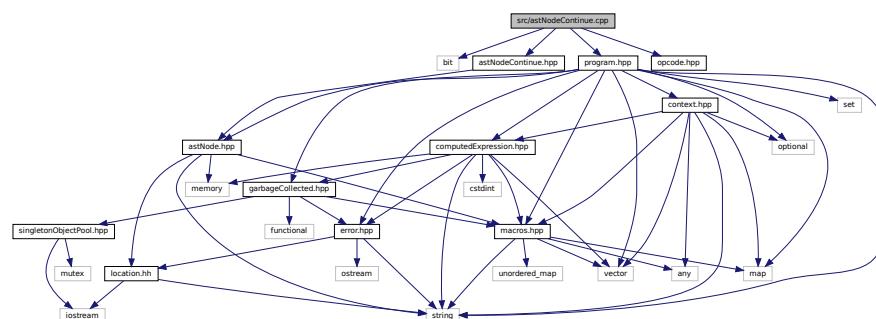
Define the `Tang::AstNodeCast` class.

6.70 src/astNodeContinue.cpp File Reference

Define the `Tang::AstNodeContinue` class.

```
#include <bit>
#include "astNodeContinue.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeContinue.cpp:
```

Include dependency graph for astNodeContinue.cpp:



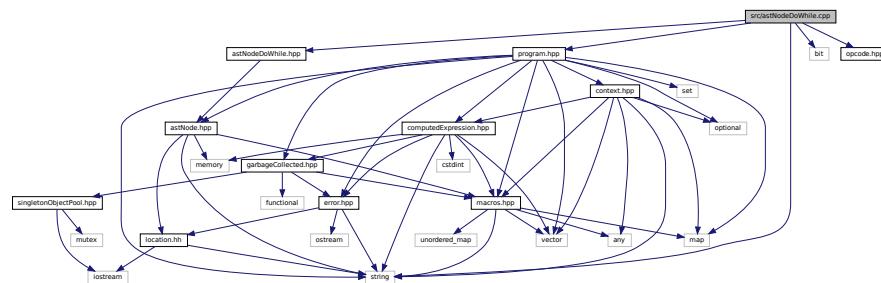
6.70.1 Detailed Description

Define the `Tang::AstNodeContinue` class.

6.71 src/astNodeDoWhile.cpp File Reference

Define the `Tang::AstNodeDoWhile` class.

```
#include <string>
#include <bit>
#include "astNodeDoWhile.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeDoWhile.cpp:
```



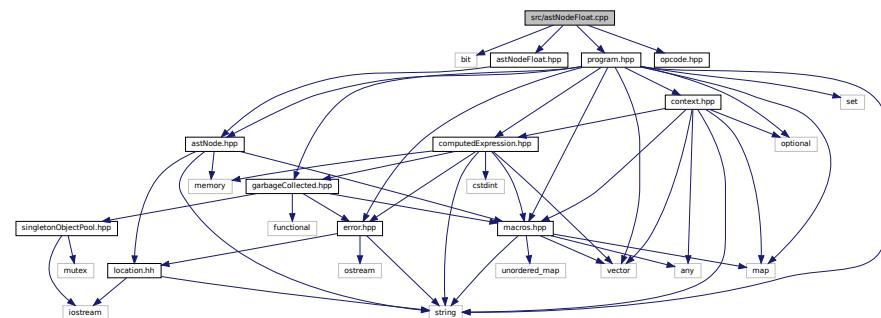
6.71.1 Detailed Description

Define the `Tang::AstNodeDoWhile` class.

6.72 src/astNodeFloat.cpp File Reference

Define the `Tang::AstNodeFloat` class.

```
#include <bit>
#include "astNodeFloat.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeFloat.hpp:
```



6.72.1 Detailed Description

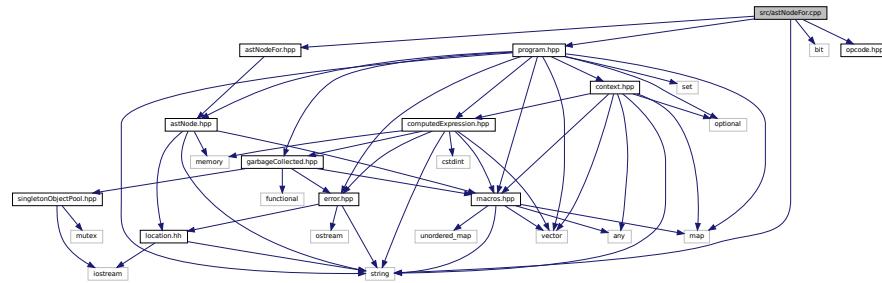
Define the `Tang::AstNodeFloat` class.

6.73 src/astNodeFor.cpp File Reference

Define the `Tang::AstNodeFor` class.

```
#include <string>
#include <bit>
#include "astNodeFor.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeFor
```

Include dependency graph for astNodeFor.cpp:



6.73.1 Detailed Description

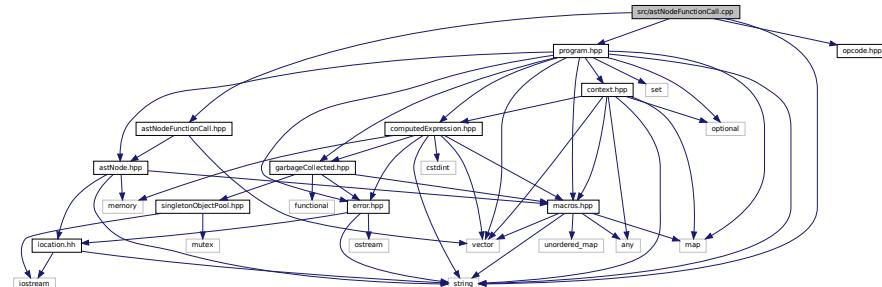
Define the `Tang::AstNodeFor` class.

6.74 src/astNodeFunctionCall.cpp File Reference

Define the `Tang::AstNodeFunctionCall` class.

```
#include <string>
#include "astNodeFunctionCall.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeFunctionCall.cpp:
```

Include dependency graph for astNodeFunctionCall.cpp:



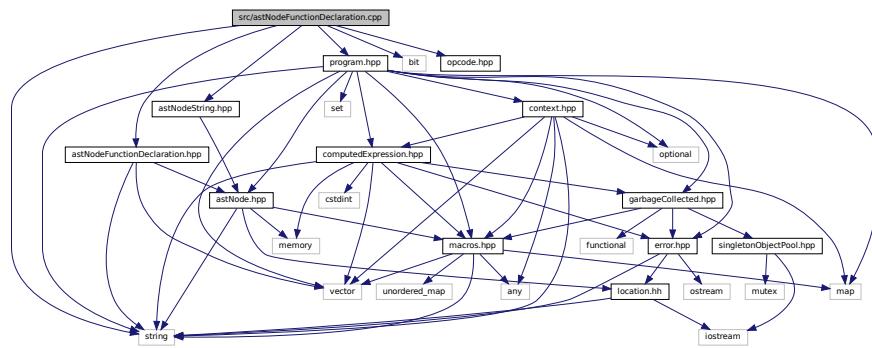
6.74.1 Detailed Description

Define the [Tang::AstNodeFunctionCall](#) class.

6.75 src/astNodeFunctionDeclaration.cpp File Reference

Define the [Tang::AstNodeFunctionDeclaration](#) class.

```
#include <string>
#include <bit>
#include "astNodeFunctionDeclaration.hpp"
#include "astNodeString.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeFunctionDeclaration.cpp:
```



6.75.1 Detailed Description

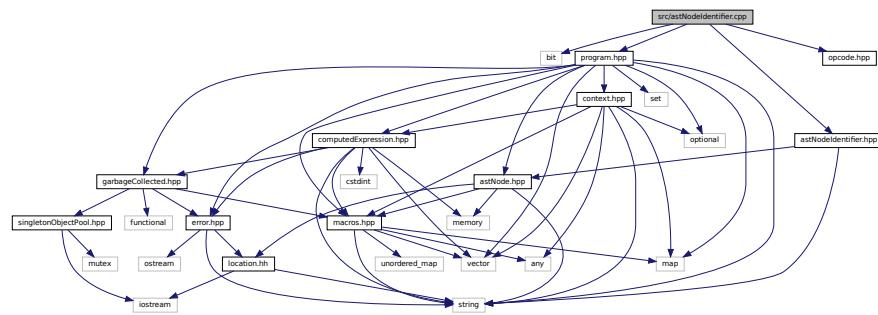
Define the [Tang::AstNodeFunctionDeclaration](#) class.

6.76 src/astNodeIdentifier.cpp File Reference

Define the [Tang::AstNodeIdentifier](#) class.

```
#include <bit>
#include "astNodeIdentifier.hpp"
#include "opcode.hpp"
```

```
#include "program.hpp"
Include dependency graph for astNodeldentifier.cpp:
```



6.76.1 Detailed Description

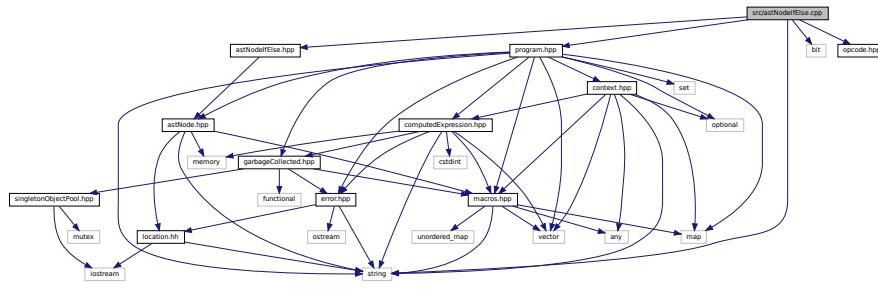
Define the `Tang::AstNodeIdentifier` class.

6.77 src/astNodeIfElse.cpp File Reference

Define the `Tang::AstNodeIfElse` class.

```
#include <string>
#include <bit>
#include "astNodeIfElse.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeIfElse.cpp:
```

Include dependency graph for astNodelfElse.cpp:



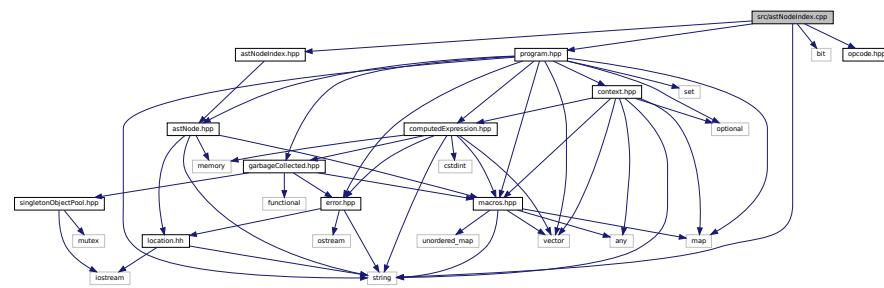
6.77.1 Detailed Description

Define the `Tang::AstNodeIfElse` class.

6.78 src/astNodeIndex.cpp File Reference

Define the `Tang::AstNodeIndex` class.

```
#include <string>
#include <bit>
#include "astNodeIndex.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeIndex.cpp:
```



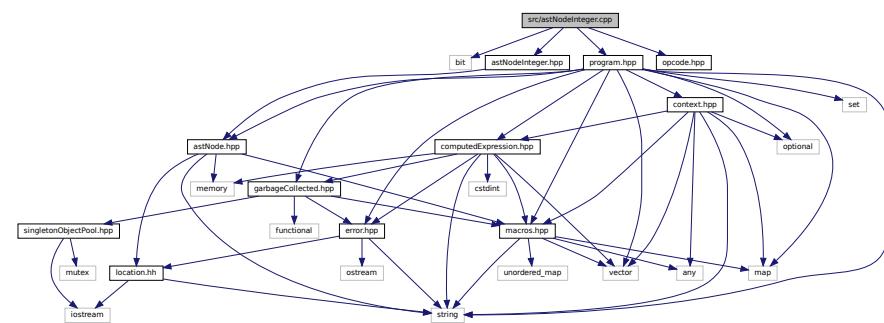
6.78.1 Detailed Description

Define the `Tang::AstNodeIndex` class.

6.79 src/astNodeInteger.cpp File Reference

Define the `Tang::AstNodeInteger` class.

```
#include <bit>
#include "astNodeInteger.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeInteger.cpp:
```



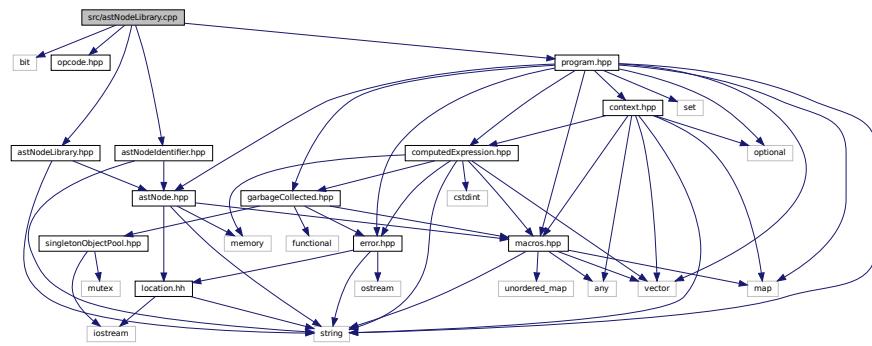
6.79.1 Detailed Description

Define the `Tang::AstNodeInteger` class.

6.80 src/astNodeLibrary.cpp File Reference

Define the `Tang::AstNodeLibrary` class.

```
#include <bit>
#include "opcode.hpp"
#include "astNodeLibrary.hpp"
#include "astNodeIdentifier.hpp"
#include "program.hpp"
Include dependency graph for astNodeLibrary.cpp:
```



6.80.1 Detailed Description

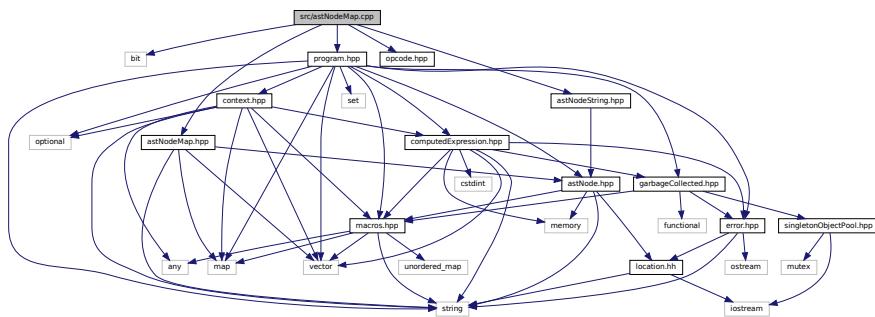
Define the `Tang::AstNodeLibrary` class.

6.81 src/astNodeMap.cpp File Reference

Define the `Tang::AstNodeMap` class.

```
#include <bit>
#include "astNodeMap.hpp"
#include "astNodeString.hpp"
#include "opcode.hpp"
```

```
#include "program.hpp"
Include dependency graph for astNodeMap.cpp:
```



6.81.1 Detailed Description

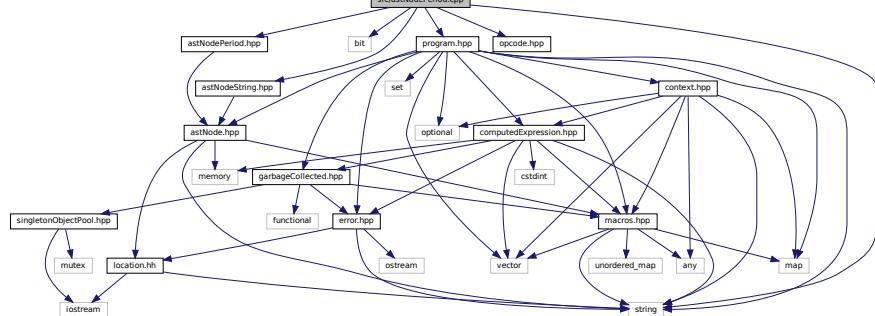
Define the `Tang::AstNodeMap` class.

6.82 src/astNodePeriod.cpp File Reference

Define the `Tang::AstNodePeriod` class.

```
#include <string>
#include <bit>
#include "astNodePeriod.hpp"
#include "astNodeString.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodePeriod.cpp:
```

include dependency graph for activated objects.



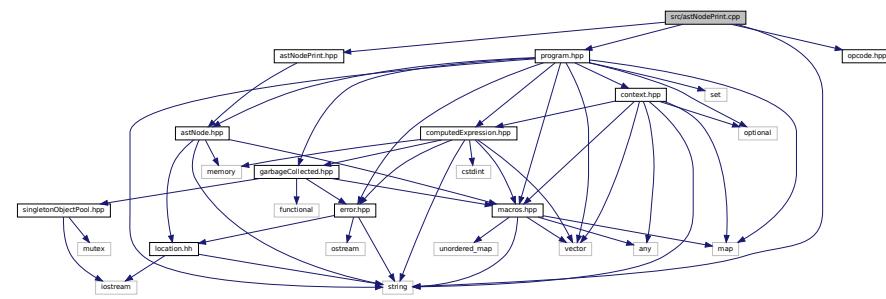
6.82.1 Detailed Description

Define the `Tang::AstNodePeriod` class.

6.83 src/astNodePrint.cpp File Reference

Define the `Tang::AstNodePrint` class.

```
#include <string>
#include "astNodePrint.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodePrint.cpp:
```



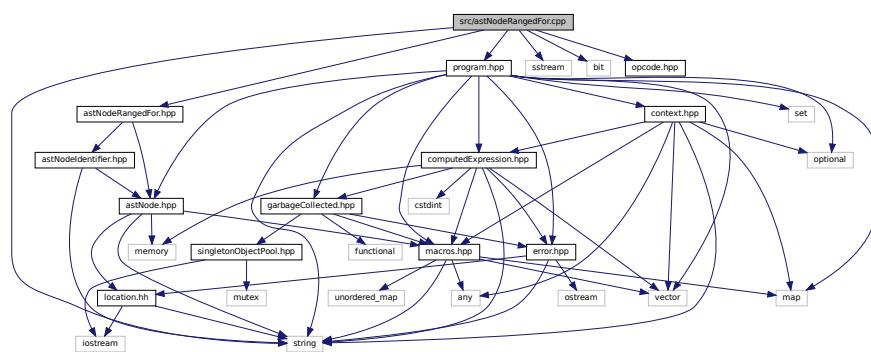
6.83.1 Detailed Description

Define the `Tang::AstNodePrint` class.

6.84 src/astNodeRangedFor.cpp File Reference

Define the `Tang::AstNodeRangedFor` class.

```
#include <string>
#include <sstream>
#include <bit>
#include "astNodeRangedFor.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeRangedFor.cpp:
```



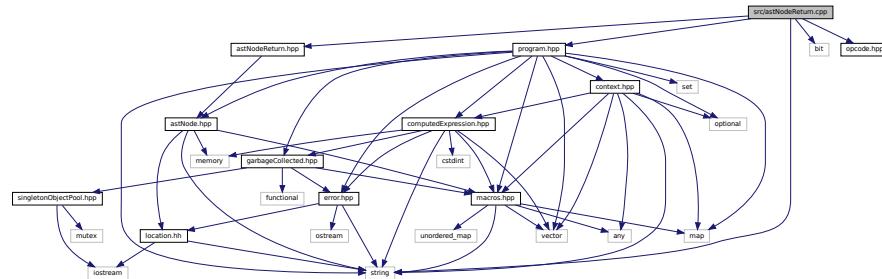
6.84.1 Detailed Description

Define the `Tang::AstNodeRangedFor` class.

6.85 src/astNodeReturn.cpp File Reference

Define the `Tang::AstNodeReturn` class.

```
#include <string>
#include <bit>
#include "astNodeReturn.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeReturn.cpp:
```



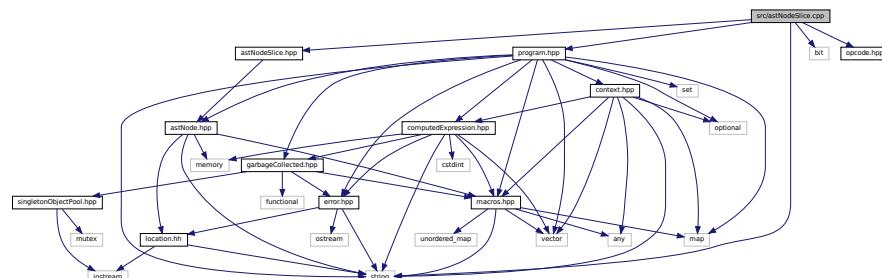
6.85.1 Detailed Description

Define the `Tang::AstNodeReturn` class.

6.86 src/astNodeSlice.cpp File Reference

Define the `Tang::AstNodeSlice` class.

```
#include <string>
#include <bit>
#include "astNodeSlice.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeSlice.cpp:
```



6.86.1 Detailed Description

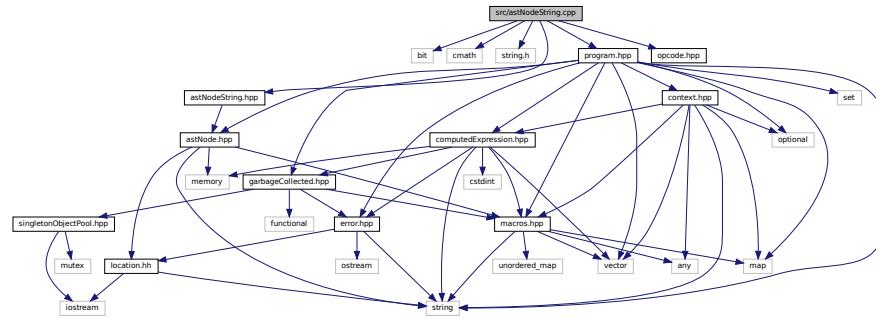
Define the `Tang::AstNodeSlice` class.

6.87 src/astNodeString.cpp File Reference

Define the `Tang::AstNodeString` class.

```
#include <bit>
#include <cmath>
#include <string.h>
#include "astNodeString.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeString.cpp:
```

Include dependency graph for astNodeString.cpp:



6.87.1 Detailed Description

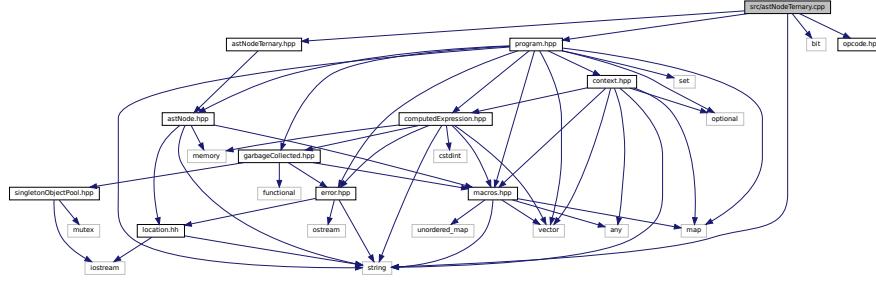
Define the `Tang::AstNodeString` class.

6.88 src/astNodeTernary.cpp File Reference

Define the `Tang::AstNodeTernary` class.

```
#include <string>
#include <bit>
#include "astNodeTernary.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeTernary.cpp:
```

Include dependency graph for astNodeTernary.cpp:



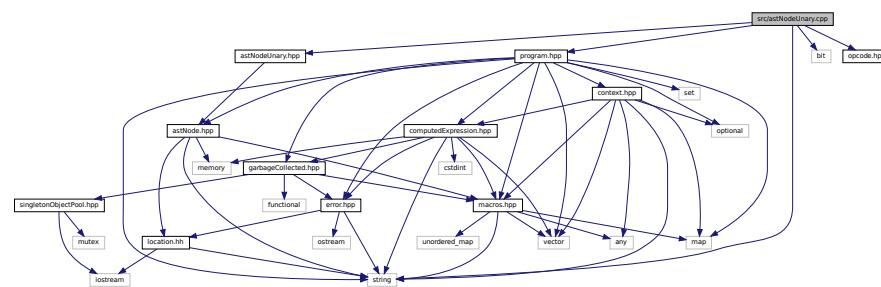
6.88.1 Detailed Description

Define the [Tang::AstNodeTernary](#) class.

6.89 src/astNodeUnary.cpp File Reference

Define the [Tang::AstNodeUnary](#) class.

```
#include <string>
#include <bit>
#include "astNodeUnary.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeUnary.cpp:
```



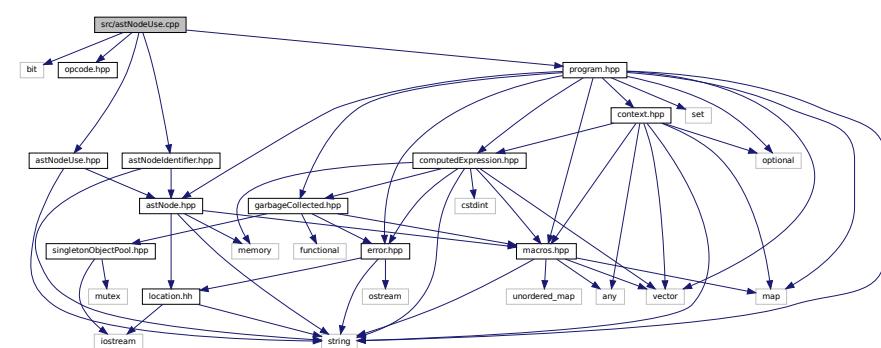
6.89.1 Detailed Description

Define the [Tang::AstNodeUnary](#) class.

6.90 src/astNodeUse.cpp File Reference

Define the [Tang::AstNodeUse](#) class.

```
#include <bit>
#include "opcode.hpp"
#include "astNodeUse.hpp"
#include "astNodeIdentifier.hpp"
#include "program.hpp"
Include dependency graph for astNodeUse.cpp:
```



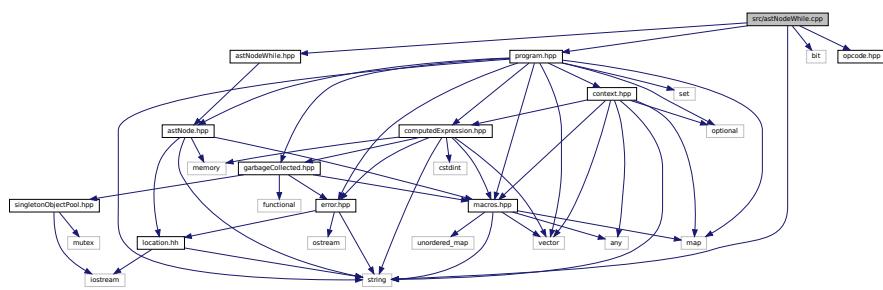
6.90.1 Detailed Description

Define the `Tang::AstNodeUse` class.

6.91 src/astNodeWhile.cpp File Reference

Define the `Tang::AstNodeWhile` class.

```
#include <string>
#include <bit>
#include "astNodeWhile.hpp"
#include "opcode.hpp"
#include "program.hpp"
Include dependency graph for astNodeWhile.cpp:
```



6.91.1 Detailed Description

Define the `Tang::AstNodeWhile` class.

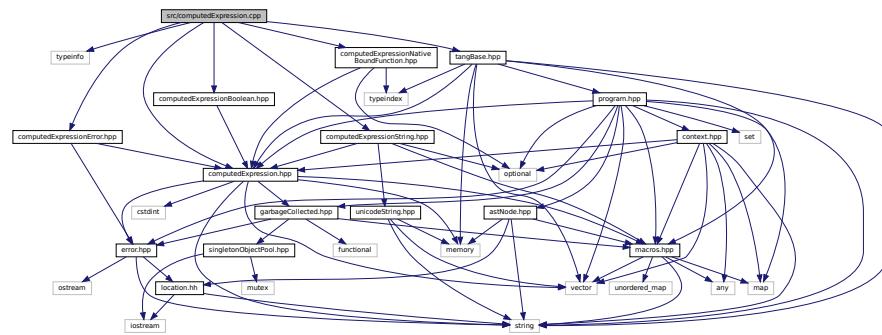
6.92 src/computedExpression.cpp File Reference

Define the `Tang::ComputedExpression` class.

```
#include <typeinfo>
#include "computedExpression.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionNativeBoundFunction.hpp"
#include "computedExpressionError.hpp"
```

```
#include "tangBase.hpp"
```

Include dependency graph for computedExpression.cpp:



6.92.1 Detailed Description

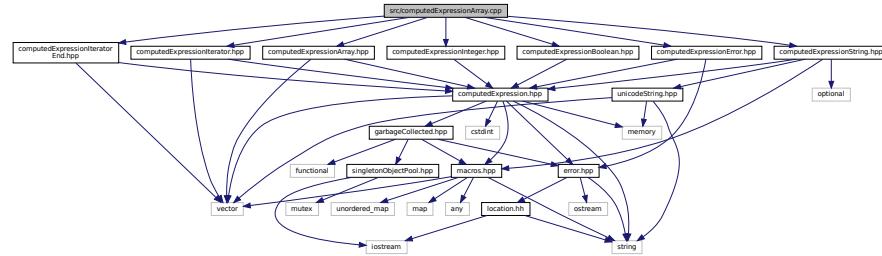
Define the `Tang::ComputedExpression` class.

6.93 src/computedExpressionArray.cpp File Reference

Define the `Tang::ComputedExpressionArray` class.

```
#include "computedExpressionArray.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionIterator.hpp"
#include "computedExpressionIteratorEnd.hpp"
#include "computedExpressionError.hpp"
Include dependency graph for computedExpressionArray.cpp:
```

Include dependency graph for computedExpressionArray.cpp:



6.93.1 Detailed Description

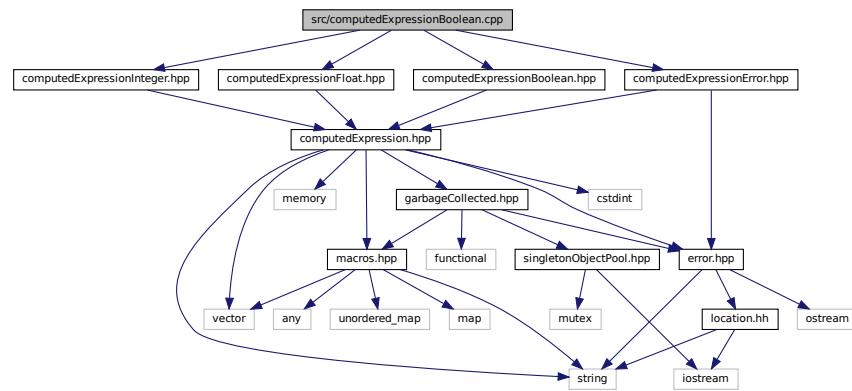
Define the `Tang::ComputedExpressionArray` class.

6.94 src/computedExpressionBoolean.cpp File Reference

Define the [Tang::ComputedExpressionBoolean](#) class.

```
#include "computedExpressionBoolean.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionError.hpp"
```

Include dependency graph for `computedExpressionBoolean.cpp`:



6.94.1 Detailed Description

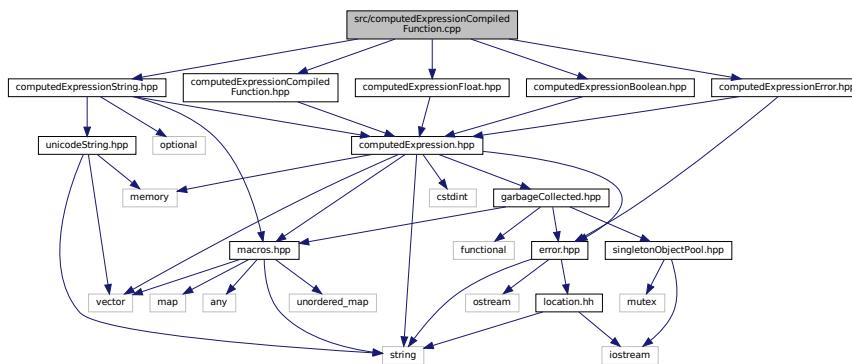
Define the [Tang::ComputedExpressionBoolean](#) class.

6.95 src/computedExpressionCompiledFunction.cpp File Reference

Define the [Tang::ComputedExpressionCompiledFunction](#) class.

```
#include "computedExpressionCompiledFunction.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionError.hpp"
```

Include dependency graph for `computedExpressionCompiledFunction.cpp`:



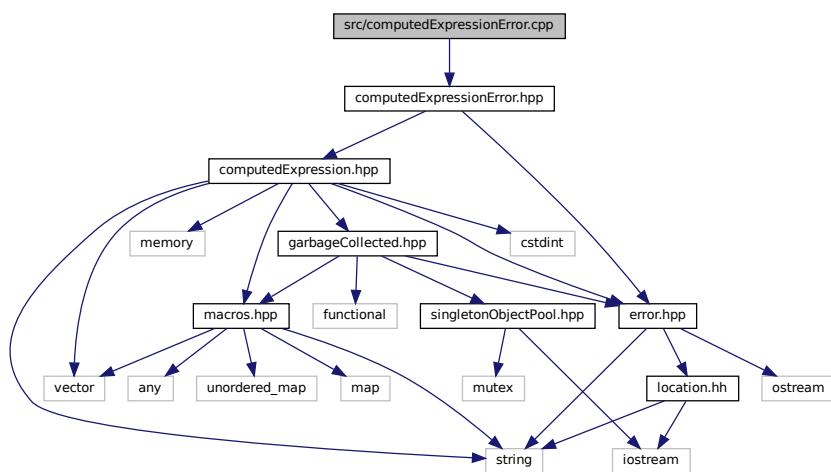
6.95.1 Detailed Description

Define the [Tang::ComputedExpressionCompiledFunction](#) class.

6.96 src/computedExpressionError.cpp File Reference

Define the [Tang::ComputedExpressionError](#) class.

```
#include "computedExpressionError.hpp"
Include dependency graph for computedExpressionError.cpp:
```



6.96.1 Detailed Description

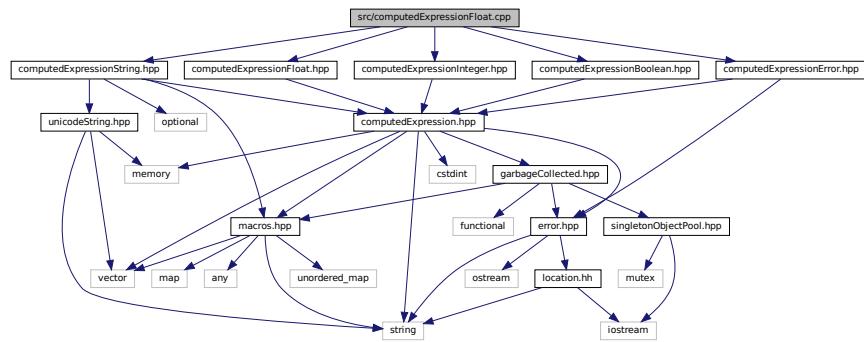
Define the [Tang::ComputedExpressionError](#) class.

6.97 src/computedExpressionFloat.cpp File Reference

Define the [Tang::ComputedExpressionFloat](#) class.

```
#include "computedExpressionFloat.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
```

```
#include "computedExpressionError.hpp"
Include dependency graph for computedExpressionFloat.cpp:
```



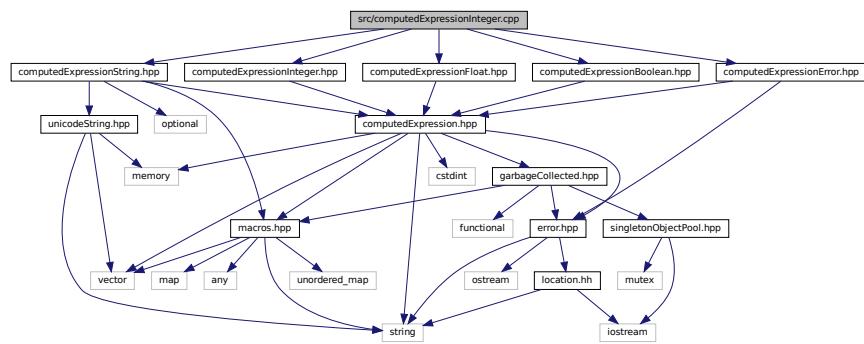
6.97.1 Detailed Description

Define the [Tang::ComputedExpressionFloat](#) class.

6.98 src/computedExpressionInteger.cpp File Reference

Define the [Tang::ComputedExpressionInteger](#) class.

```
#include "computedExpressionInteger.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionError.hpp"
Include dependency graph for computedExpressionInteger.cpp:
```



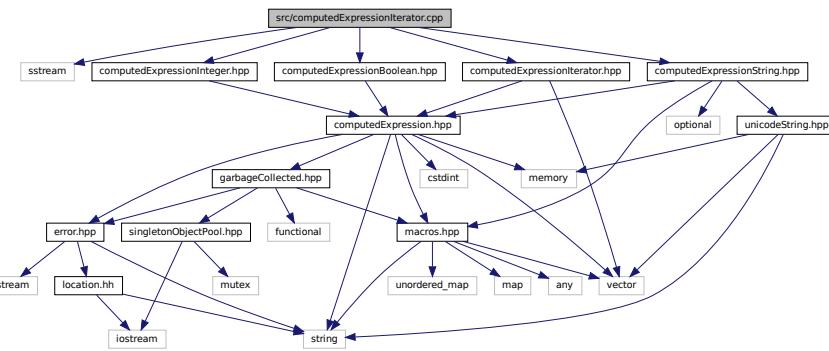
6.98.1 Detailed Description

Define the [Tang::ComputedExpressionInteger](#) class.

6.99 src/computedExpressionIterator.cpp File Reference

Define the [Tang::ComputedExpressionIterator](#) class.

```
#include <sstream>
#include "computedExpressionIterator.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
Include dependency graph for computedExpressionIterator.cpp:
```



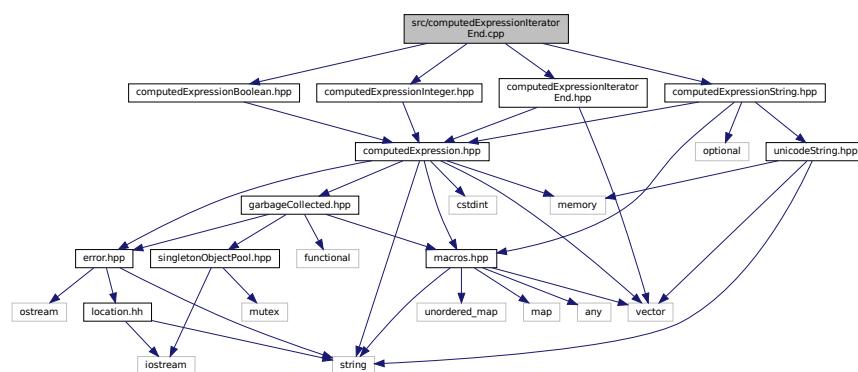
6.99.1 Detailed Description

Define the [Tang::ComputedExpressionIterator](#) class.

6.100 src/computedExpressionIteratorEnd.cpp File Reference

Define the [Tang::ComputedExpressionIteratorEnd](#) class.

```
#include "computedExpressionIteratorEnd.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
Include dependency graph for computedExpressionIteratorEnd.cpp:
```



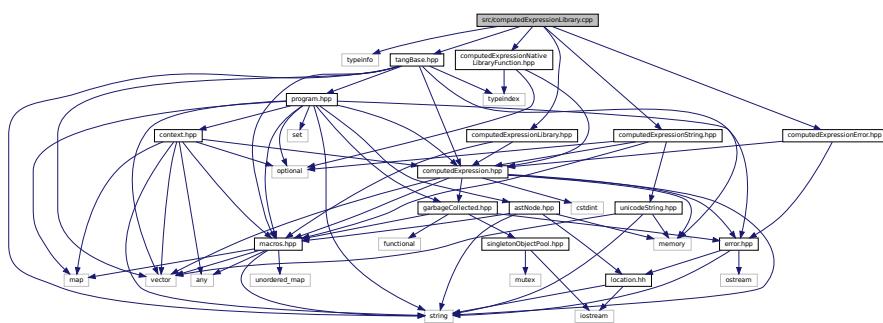
6.100.1 Detailed Description

Define the `Tang::ComputedExpressionIteratorEnd` class.

6.101 src/computedExpressionLibrary.cpp File Reference

Define the `Tang::ComputedExpressionLibrary` class.

```
#include <typeinfo>
#include "tangBase.hpp"
#include "computedExpressionError.hpp"
#include "computedExpressionLibrary.hpp"
#include "computedExpressionNativeLibraryFunction.hpp"
#include "computedExpressionString.hpp"
Include dependency graph for computedExpressionLibrary.cpp:
```



6.101.1 Detailed Description

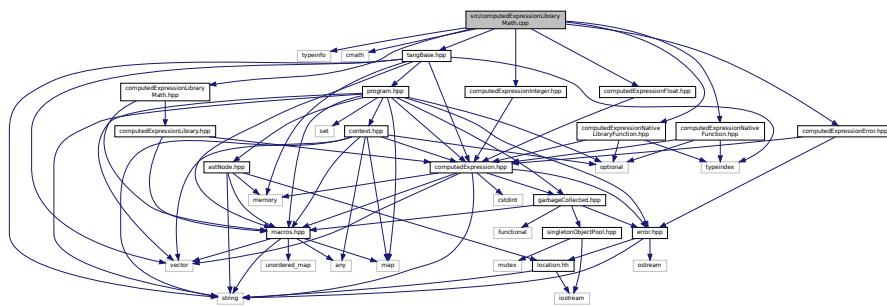
Define the `Tang::ComputedExpressionLibrary` class.

6.102 src/computedExpressionLibraryMath.cpp File Reference

Define the `Tang::ComputedExpressionLibraryMath` class.

```
#include <typeinfo>
#include <cmath>
#include "tangBase.hpp"
#include "computedExpressionError.hpp"
#include "computedExpressionLibraryMath.hpp"
#include "computedExpressionNativeLibraryFunction.hpp"
#include "computedExpressionNativeFunction.hpp"
#include "computedExpressionInteger.hpp"
```

```
#include "computedExpressionFloat.hpp"
Include dependency graph for computedExpressionLibraryMath.cpp:
```



6.102.1 Detailed Description

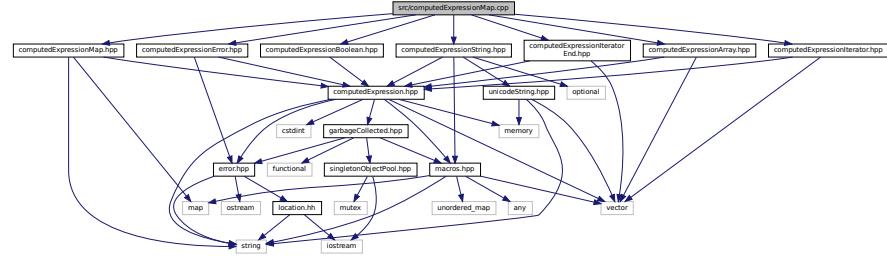
Define the `Tang::ComputedExpressionLibraryMath` class.

6.103 src/computedExpressionMap.cpp File Reference

Define the `Tang::ComputedExpressionMap` class.

```
#include "computedExpressionMap.hpp"
#include "computedExpressionArray.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionIterator.hpp"
#include "computedExpressionIteratorEnd.hpp"
#include "computedExpressionError.hpp"
Include dependency graph for computedExpressionMap.cpp:
```

Include dependency graph for computedExpressionMap.cpp:



6.103.1 Detailed Description

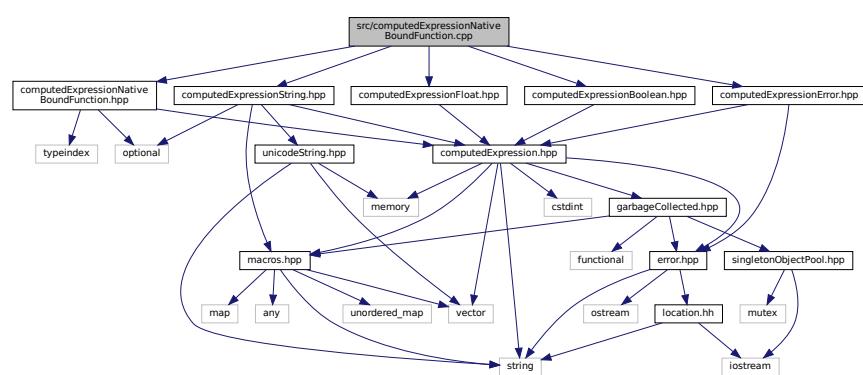
Define the `Tang::ComputedExpressionMap` class.

6.104 src/computedExpressionNativeBoundFunction.cpp File Reference

Define the [Tang::ComputedExpressionNativeBoundFunction](#) class.

```
#include "computedExpressionNativeBoundFunction.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionError.hpp"
```

Include dependency graph for `computedExpressionNativeBoundFunction.cpp`:



6.104.1 Detailed Description

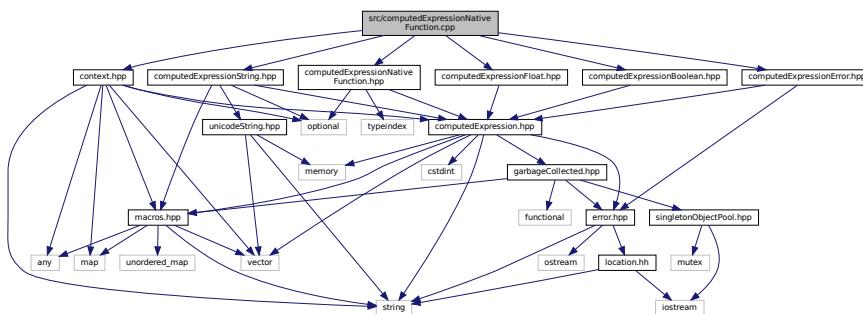
Define the [Tang::ComputedExpressionNativeBoundFunction](#) class.

6.105 src/computedExpressionNativeFunction.cpp File Reference

Define the [Tang::ComputedExpressionNativeFunction](#) class.

```
#include "context.hpp"
#include "computedExpressionNativeFunction.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionError.hpp"
```

Include dependency graph for `computedExpressionNativeFunction.cpp`:



6.105.1 Detailed Description

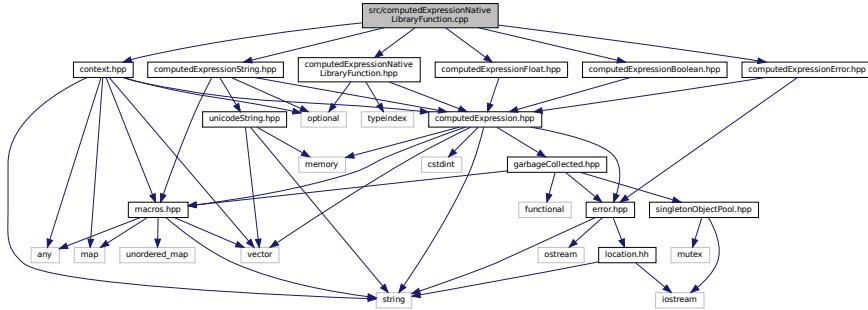
Define the [Tang::ComputedExpressionNativeFunction](#) class.

6.106 src/computedExpressionNativeLibraryFunction.cpp File Reference

Define the [Tang::ComputedExpressionNativeLibraryFunction](#) class.

```
#include "context.hpp"
#include "computedExpressionNativeLibraryFunction.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionError.hpp"
```

Include dependency graph for `computedExpressionNativeLibraryFunction.cpp`:



6.106.1 Detailed Description

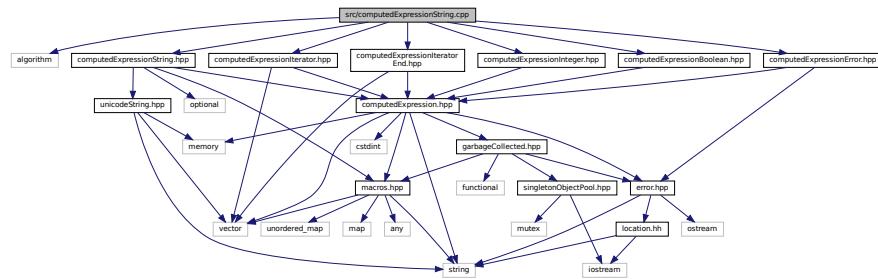
Define the [Tang::ComputedExpressionNativeLibraryFunction](#) class.

6.107 src/computedExpressionString.cpp File Reference

Define the [Tang::ComputedExpressionString](#) class.

```
#include <algorithm>
#include "computedExpressionString.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionError.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionIterator.hpp"
```

```
#include "computedExpressionIteratorEnd.hpp"
Include dependency graph for computedExpressionString.cpp:
```



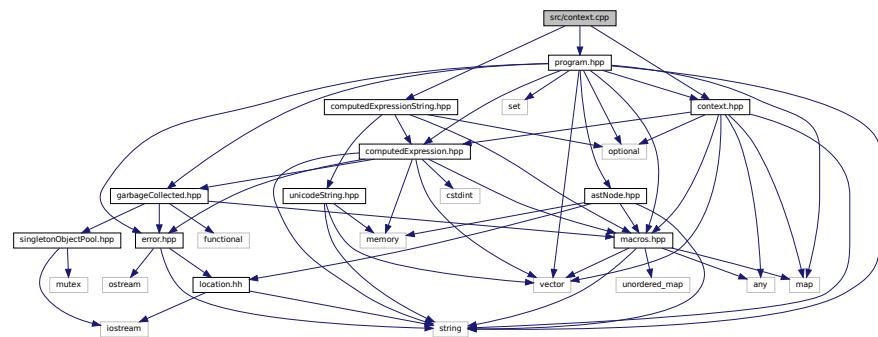
6.107.1 Detailed Description

Define the [Tang::ComputedExpressionString](#) class.

6.108 src/context.cpp File Reference

Define the [Tang::Context](#) class.

```
#include "context.hpp"
#include "program.hpp"
#include "computedExpressionString.hpp"
Include dependency graph for context.cpp:
```



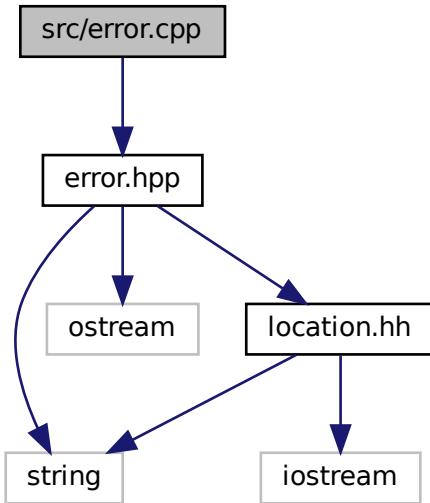
6.108.1 Detailed Description

Define the [Tang::Context](#) class.

6.109 src/error.cpp File Reference

Define the [Tang::Error](#) class.

```
#include "error.hpp"
Include dependency graph for error.cpp:
```



Functions

- `std::ostream & Tang::operator<< (std::ostream &out, const Error &error)`

6.109.1 Detailed Description

Define the [Tang::Error](#) class.

6.109.2 Function Documentation

6.109.2.1 operator<<()

```
std::ostream& Tang::operator<< (
    std::ostream & out,
    const Error & error )
```

Parameters

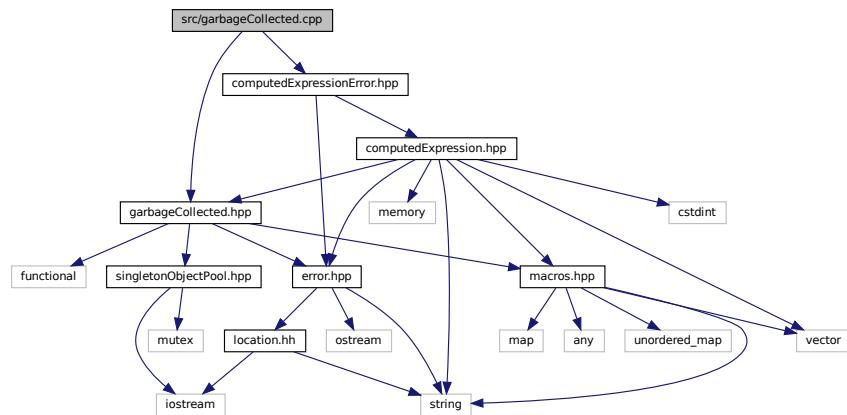
<i>out</i>	The output stream.
<i>error</i>	The Error object.

Returns

The output stream.

6.110 src/garbageCollected.cpp File Reference

```
#include "garbageCollected.hpp"
#include "computedExpressionError.hpp"
Include dependency graph for garbageCollected.cpp:
```



Functions

- std::ostream & [Tang::operator<<](#) (std::ostream &out, const [GarbageCollected](#) &gc)

6.110.1 Function Documentation

6.110.1.1 operator<<()

```
std::ostream& Tang::operator<< (
    std::ostream & out,
    const GarbageCollected & gc )
```

Parameters

<i>out</i>	The output stream.
<i>gc</i>	The GarbageCollected value.

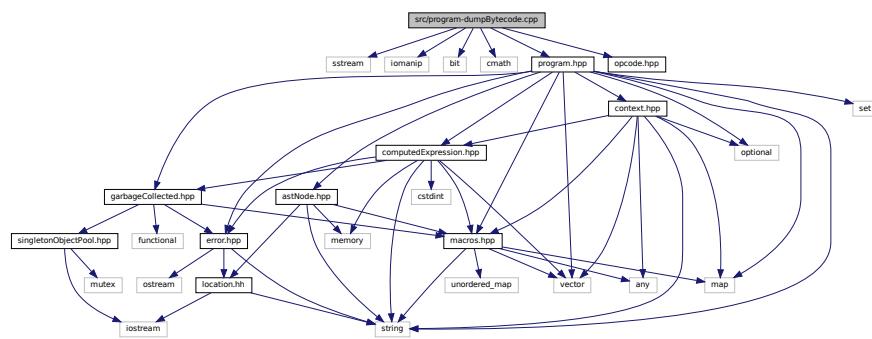
Returns

The output stream.

6.111 src/program-dumpBytecode.cpp File Reference

Define the `Tang::Program::dumpBytecode` method.

```
#include <sstream>
#include <iomanip>
#include <bit>
#include <cmath>
#include "program.hpp"
#include "opcode.hpp"
```



Macros

- `#define DUMPPROGRAMCHECK(x)`

Verify the size of the Bytecode vector so that it may be safely accessed.

6.111.1 Detailed Description

Define the `Tang::Program::dumpBytecode` method.

6.111.2 Macro Definition Documentation

6.111.2.1 DUMPPROGRAMCHECK

```
#define DUMPPROGRAMCHECK( x )
```

Value:

```
if (this->bytecode.size() < (pc + (x))) \
    return out.str() + "Error: Opcode truncated\n"
```

Verify the size of the Bytecode vector so that it may be safely accessed.

If the vector is not large enough, an error message is appended to the output string and no further opcodes are printed.

Parameters

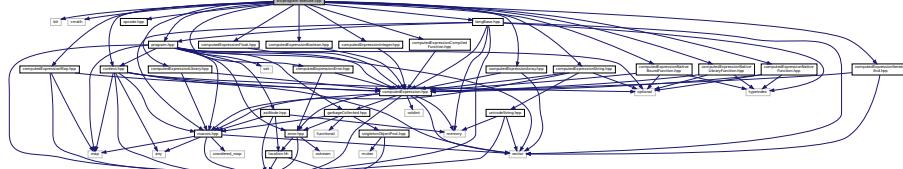
x	The number of additional vector entries that should exist.
---	--

6.112 src/program-execute.cpp File Reference

Define the [Tang::Program::execute](#) method.

```
#include <bit>
#include <cmath>
#include "program.hpp"
#include "context.hpp"
#include "opcode.hpp"
#include "computedExpressionError.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionArray.hpp"
#include "computedExpressionLibrary.hpp"
#include "computedExpressionMap.hpp"
#include "computedExpressionCompiledFunction.hpp"
#include "computedExpressionNativeBoundFunction.hpp"
#include "computedExpressionNativeLibraryFunction.hpp"
#include "computedExpressionNativeFunction.hpp"
#include "computedExpressionIteratorEnd.hpp"
#include "tangBase.hpp"
```

Include dependency graph for program-execute.cpp:



Macros

- `#define EXECUTEPROGRAMCHECK(x)`
Verify the size of the Bytecode vector so that it may be safely accessed.
- `#define STACKCHECK(x)`
Verify the size of the stack vector so that it may be safely accessed.

6.112.1 Detailed Description

Define the `Tang::Program::execute` method.

6.112.2 Macro Definition Documentation

6.112.2.1 EXECUTEPROGRAMCHECK

```
#define EXECUTEPROGRAMCHECK(
    x )
```

Value:

```
if (this->bytecode.size() < (pc + (x))) { \
    stack.push_back(GarbageCollected::make<ComputedExpressionError>(Error{"Opcode instruction \
truncated."})); \
    pc = this->bytecode.size(); \
    break; \
}
```

Verify the size of the Bytecode vector so that it may be safely accessed.

Parameters

x	The number of additional vector entries that should exist.
---	--

6.112.2.2 STACKCHECK

```
#define STACKCHECK(
    x )
```

Value:

```
if (stack.size() < (fp + (x))) { \
    stack.push_back(GarbageCollected::make<ComputedExpressionError>(Error{"Insufficient stack depth."})); \
    pc = this->bytecode.size(); \
    break; \
}
```

Verify the size of the stack vector so that it may be safely accessed.

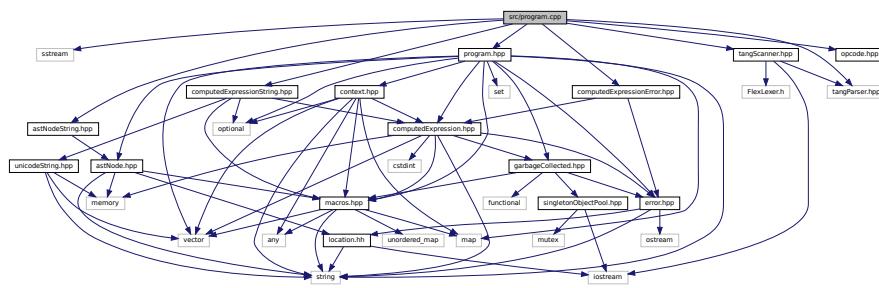
Parameters

x	The number of entries that should exist in the stack.
---	---

6.113 src/program.cpp File Reference

Define the [Tang::Program](#) class.

```
#include <sstream>
#include "program.hpp"
#include "opcode.hpp"
#include "tangScanner.hpp"
#include "tangParser.hpp"
#include "astNodeString.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionError.hpp"
Include dependency graph for program.cpp:
```



6.113.1 Detailed Description

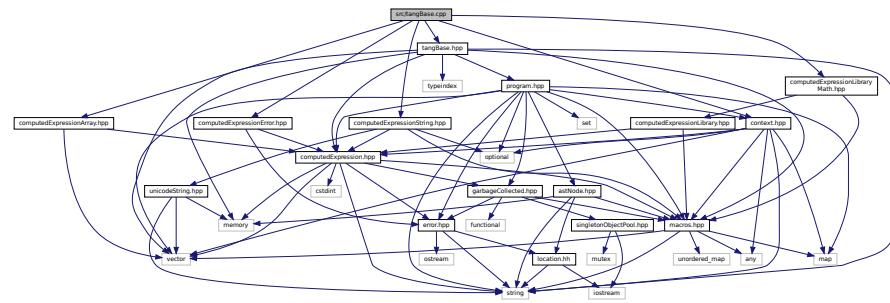
Define the [Tang::Program](#) class.

6.114 src/tangBase.cpp File Reference

Define the [Tang::TangBase](#) class.

```
#include "tangBase.hpp"
#include "computedExpressionArray.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionError.hpp"
#include "computedExpressionLibraryMath.hpp"
```

```
#include "context.hpp"
Include dependency graph for tangBase.cpp:
```



6.114.1 Detailed Description

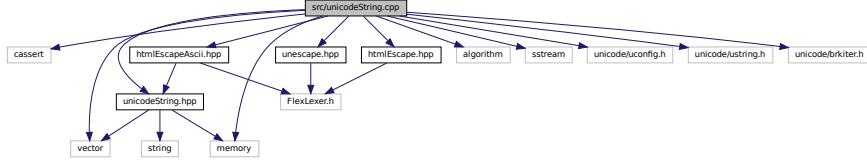
Define the `Tang::TangBase` class.

6.115 src/unicodeString.cpp File Reference

Contains the function declarations for the [Tang::UnicodeString](#) class and the interface to ICU.

```
#include <cassert>
#include <vector>
#include <memory>
#include <algorithm>
#include <sstream>
#include <unicode/uconfig.h>
#include <unicode/ustring.h>
#include <unicode/brkiter.h>
#include "unicodeString.hpp"
#include "unescape.hpp"
#include "htmlEscape.hpp"
#include "htmlEscapeAscii.hpp"
Include dependency graph for unicodeString.cpp:
```

Include dependency graph for `unhexString.cpp`:



6.115.1 Detailed Description

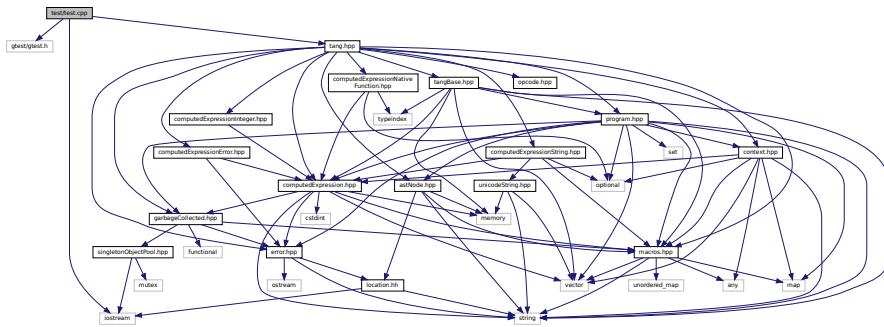
Contains the function declarations for the [Tang::UnicodeString](#) class and the interface to ICU.

6.116 test/test.cpp File Reference

Test the general language behaviors.

```
#include <gtest/gtest.h>
#include <iostream>
#include "tang.hpp"
```

Include dependency graph for test.cpp:



Functions

- **TEST** (Declare, Null)
 - **TEST** (Declare, Integer)
 - **TEST** (Declare, Float)
 - **TEST** (Declare, Boolean)
 - **TEST** (Declare, String)
 - **TEST** (Expression, Add)
 - **TEST** (Expression, Subtract)
 - **TEST** (Expression, Multiplication)
 - **TEST** (Expression, Division)
 - **TEST** (Expression, Modulo)
 - **TEST** (Expression, UnaryMinus)
 - **TEST** (Expression, Parentheses)
 - **TEST** (Expression, TypeCast)
 - **TEST** (Expression, Not)
 - **TEST** (Expression, LessThan)
 - **TEST** (Expression, LessThanEqual)
 - **TEST** (Expression, GreaterThan)
 - **TEST** (Expression, GreaterThanEqual)
 - **TEST** (Expression, Equal)
 - **TEST** (Expression, NotEqual)
 - **TEST** (Expression, And)
 - **TEST** (Expression, Or)
 - **TEST** (Expression, Ternary)
 - **TEST** (Expression, StringIndex)
 - **TEST** (Expression, StringSlice)
 - **TEST** (Expression, ArrayIndex)
 - **TEST** (Expression, Map)
 - **TEST** (CodeBlock, Statements)
 - **TEST** (Assign, Identifier)
 - **TEST** (Assign, Index)

- **TEST** (Expression, ArraySlice)
- **TEST** (ControlFlow, IfElse)
- **TEST** (ControlFlow, While)
- **TEST** (ControlFlow, Break)
- **TEST** (ControlFlow, Continue)
- **TEST** (ControlFlow, DoWhile)
- **TEST** (ControlFlow, For)
- **TEST** (ControlFlow, RangedFor)
- **TEST** (Print, Default)
- **TEST** (Print, Array)
- **TEST** (Syntax, SingleLineComment)
- **TEST** (Syntax, MultiLineComment)
- **TEST** (Syntax, UntrustedString)
- **TEST** (Syntax, UntrustedStringLiteral)
- **TEST** (NativeFunctions, General)
- **TEST** ([Context](#), General)
- **TEST** (Compile, Template)
- **TEST** (Compile, ShortCodes)
- **TEST** (Library, Use)
- int **main** (int argc, char **argv)

Variables

- auto **tang** = TangBase::make_shared()

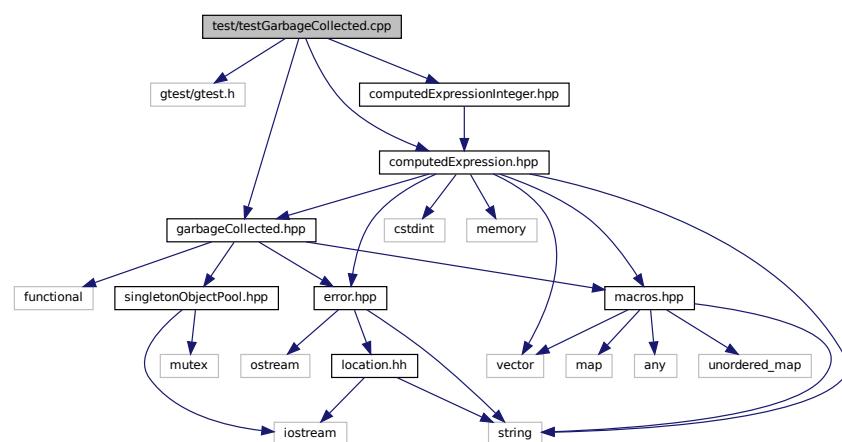
6.116.1 Detailed Description

Test the general language behaviors.

6.117 test/testGarbageCollected.cpp File Reference

Test the generic behavior of the [Tang::GarbageCollected](#) class.

```
#include <gtest/gtest.h>
#include "garbageCollected.hpp"
#include "computedExpression.hpp"
#include "computedExpressionInteger.hpp"
Include dependency graph for testGarbageCollected.cpp:
```



Functions

- **TEST** (Create, Access)
- **TEST** (RuleOfFive, CopyConstructor)
- **TEST** (Recycle, ObjectIsRecycled)
- **TEST** (Recycle, ObjectIsNotRecycled)
- int **main** (int argc, char **argv)

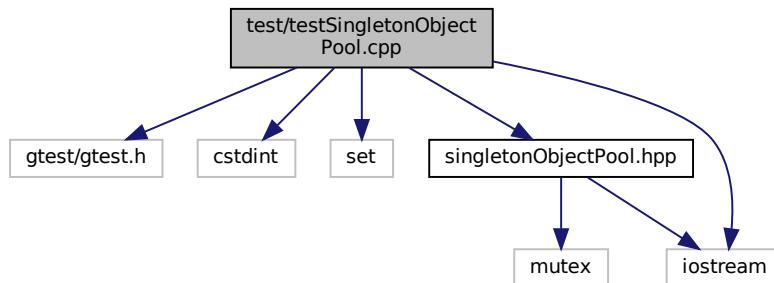
6.117.1 Detailed Description

Test the generic behavior of the [Tang::GarbageCollected](#) class.

6.118 test/testSingletonObjectPool.cpp File Reference

Test the generic behavior of the [Tang::SingletonObjectPool](#) class.

```
#include <gtest/gtest.h>
#include <cstdint>
#include <set>
#include "singletonObjectPool.hpp"
#include <iostream>
Include dependency graph for testSingletonObjectPool.cpp:
```



Functions

- **TEST** (Singleton, SameForSameType)
- **TEST** (Singleton, DifferentForDifferentTypes)
- **TEST** (Get, SuccessiveCallsProduceDifferentMemoryAddresses)
- **TEST** (Recycle, RecycledObjectIsReused)
- **TEST** (Get, SuccessiveCallsAreSequential)
- **TEST** (Get, KeepsGeneratingDifferentPointers)
- **TEST** (Recycle, WorksAfterLargeNumberOfAllocations)
- int **main** (int argc, char **argv)

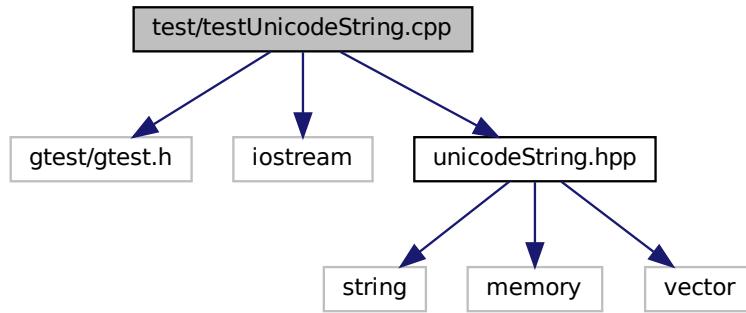
6.118.1 Detailed Description

Test the generic behavior of the [Tang::SingletonObjectPool](#) class.

6.119 test/testUnicodeString.cpp File Reference

Contains tests for the [Tang::UnicodeString](#) class.

```
#include <gtest/gtest.h>
#include <iostream>
#include "unicodeString.hpp"
Include dependency graph for testUnicodeString.cpp:
```



Functions

- [TEST \(Core, Unescape\)](#)
- [TEST \(Core, HtmlEscape\)](#)
- [TEST \(Core, HtmlEscapeAscii\)](#)
- [TEST \(UnicodeString, SubString\)](#)
- [TEST \(UnicodeString, Types\)](#)
- int `main` (int argc, char **argv)

6.119.1 Detailed Description

Contains tests for the [Tang::UnicodeString](#) class.

Index

—add
 Tang::ComputedExpression, 153
 Tang::ComputedExpressionArray, 166
 Tang::ComputedExpressionBoolean, 181
 Tang::ComputedExpressionCompiledFunction, 194
 Tang::ComputedExpressionError, 207
 Tang::ComputedExpressionFloat, 220
 Tang::ComputedExpressionInteger, 234
 Tang::ComputedExpressionIterator, 248
 Tang::ComputedExpressionIteratorEnd, 262
 Tang::ComputedExpressionLibrary, 276
 Tang::ComputedExpressionLibraryMath, 290
 Tang::ComputedExpressionMap, 304
 Tang::ComputedExpressionNativeBoundFunction, 318
 Tang::ComputedExpressionNativeFunction, 333
 Tang::ComputedExpressionNativeLibraryFunction, 348
 Tang::ComputedExpressionString, 364

—asCode
 Tang::ComputedExpression, 153
 Tang::ComputedExpressionArray, 166
 Tang::ComputedExpressionBoolean, 181
 Tang::ComputedExpressionCompiledFunction, 194
 Tang::ComputedExpressionError, 207
 Tang::ComputedExpressionFloat, 220
 Tang::ComputedExpressionInteger, 234
 Tang::ComputedExpressionIterator, 249
 Tang::ComputedExpressionIteratorEnd, 263
 Tang::ComputedExpressionLibrary, 277
 Tang::ComputedExpressionLibraryMath, 291
 Tang::ComputedExpressionMap, 304
 Tang::ComputedExpressionNativeBoundFunction, 319
 Tang::ComputedExpressionNativeFunction, 334
 Tang::ComputedExpressionNativeLibraryFunction, 349
 Tang::ComputedExpressionString, 364

—assign_index
 Tang::ComputedExpression, 153
 Tang::ComputedExpressionArray, 167
 Tang::ComputedExpressionBoolean, 182
 Tang::ComputedExpressionCompiledFunction, 194
 Tang::ComputedExpressionError, 208
 Tang::ComputedExpressionFloat, 220
 Tang::ComputedExpressionInteger, 234
 Tang::ComputedExpressionIterator, 249
 Tang::ComputedExpressionIteratorEnd, 263
 Tang::ComputedExpressionLibrary, 277

—divide
 Tang::ComputedExpression, 154
 Tang::ComputedExpressionArray, 168
 Tang::ComputedExpressionBoolean, 182
 Tang::ComputedExpressionCompiledFunction, 195
 Tang::ComputedExpressionError, 208
 Tang::ComputedExpressionFloat, 221
 Tang::ComputedExpressionInteger, 235
 Tang::ComputedExpressionIterator, 250
 Tang::ComputedExpressionIteratorEnd, 264
 Tang::ComputedExpressionLibrary, 278
 Tang::ComputedExpressionLibraryMath, 292
 Tang::ComputedExpressionMap, 305
 Tang::ComputedExpressionNativeBoundFunction, 320
 Tang::ComputedExpressionNativeFunction, 335
 Tang::ComputedExpressionNativeLibraryFunction, 350
 Tang::ComputedExpressionString, 366

—equal
 Tang::ComputedExpression, 154
 Tang::ComputedExpressionArray, 168

Tang::ComputedExpressionBoolean, 183
 Tang::ComputedExpressionCompiledFunction, 196
 Tang::ComputedExpressionError, 209
 Tang::ComputedExpressionFloat, 222
 Tang::ComputedExpressionInteger, 236
 Tang::ComputedExpressionIterator, 250
 Tang::ComputedExpressionIteratorEnd, 264
 Tang::ComputedExpressionLibrary, 278
 Tang::ComputedExpressionLibraryMath, 292
 Tang::ComputedExpressionMap, 306
 Tang::ComputedExpressionNativeBoundFunction, 320
 Tang::ComputedExpressionNativeFunction, 335
 Tang::ComputedExpressionNativeLibraryFunction, 350
 Tang::ComputedExpressionString, 366
float
 Tang::ComputedExpression, 155
 Tang::ComputedExpressionArray, 169
 Tang::ComputedExpressionBoolean, 183
 Tang::ComputedExpressionCompiledFunction, 196
 Tang::ComputedExpressionError, 209
 Tang::ComputedExpressionFloat, 222
 Tang::ComputedExpressionInteger, 236
 Tang::ComputedExpressionIterator, 251
 Tang::ComputedExpressionIteratorEnd, 265
 Tang::ComputedExpressionLibrary, 279
 Tang::ComputedExpressionLibraryMath, 293
 Tang::ComputedExpressionMap, 306
 Tang::ComputedExpressionNativeBoundFunction, 321
 Tang::ComputedExpressionNativeFunction, 336
 Tang::ComputedExpressionNativeLibraryFunction, 351
 Tang::ComputedExpressionString, 367
getIterator
 Tang::ComputedExpression, 155
 Tang::ComputedExpressionArray, 169
 Tang::ComputedExpressionBoolean, 183
 Tang::ComputedExpressionCompiledFunction, 196
 Tang::ComputedExpressionError, 209
 Tang::ComputedExpressionFloat, 223
 Tang::ComputedExpressionInteger, 237
 Tang::ComputedExpressionIterator, 251
 Tang::ComputedExpressionIteratorEnd, 265
 Tang::ComputedExpressionLibrary, 279
 Tang::ComputedExpressionLibraryMath, 293
 Tang::ComputedExpressionMap, 306
 Tang::ComputedExpressionNativeBoundFunction, 321
 Tang::ComputedExpressionNativeFunction, 336
 Tang::ComputedExpressionNativeLibraryFunction, 351
 Tang::ComputedExpressionString, 367
index
 Tang::ComputedExpression, 155
 Tang::ComputedExpressionArray, 169
 Tang::ComputedExpressionBoolean, 184
 Tang::ComputedExpressionCompiledFunction, 197
 Tang::ComputedExpressionError, 210
 Tang::ComputedExpressionFloat, 223
 Tang::ComputedExpressionInteger, 237
 Tang::ComputedExpressionIterator, 251
 Tang::ComputedExpressionIteratorEnd, 265
 Tang::ComputedExpressionLibrary, 279
 Tang::ComputedExpressionLibraryMath, 293
 Tang::ComputedExpressionMap, 307
 Tang::ComputedExpressionNativeBoundFunction, 321
 Tang::ComputedExpressionNativeFunction, 337
 Tang::ComputedExpressionNativeLibraryFunction, 351
 Tang::ComputedExpressionString, 368
integer
 Tang::ComputedExpression, 156
 Tang::ComputedExpressionArray, 170
 Tang::ComputedExpressionBoolean, 184
 Tang::ComputedExpressionCompiledFunction, 197
 Tang::ComputedExpressionError, 210
 Tang::ComputedExpressionFloat, 223
 Tang::ComputedExpressionInteger, 237
 Tang::ComputedExpressionIterator, 252
 Tang::ComputedExpressionIteratorEnd, 266
 Tang::ComputedExpressionLibrary, 280
 Tang::ComputedExpressionLibraryMath, 294
 Tang::ComputedExpressionMap, 307
 Tang::ComputedExpressionNativeBoundFunction, 322
 Tang::ComputedExpressionNativeFunction, 337
 Tang::ComputedExpressionNativeLibraryFunction, 352
 Tang::ComputedExpressionString, 368
iteratorNext
 Tang::ComputedExpression, 156
 Tang::ComputedExpressionArray, 170
 Tang::ComputedExpressionBoolean, 184
 Tang::ComputedExpressionCompiledFunction, 197
 Tang::ComputedExpressionError, 210
 Tang::ComputedExpressionFloat, 223
 Tang::ComputedExpressionInteger, 237
 Tang::ComputedExpressionIterator, 252
 Tang::ComputedExpressionIteratorEnd, 266
 Tang::ComputedExpressionLibrary, 280
 Tang::ComputedExpressionLibraryMath, 294
 Tang::ComputedExpressionMap, 307
 Tang::ComputedExpressionNativeBoundFunction, 322
 Tang::ComputedExpressionNativeFunction, 337
 Tang::ComputedExpressionNativeLibraryFunction, 352
 Tang::ComputedExpressionString, 368
lessThan
 Tang::ComputedExpression, 156
 Tang::ComputedExpressionArray, 170
 Tang::ComputedExpressionBoolean, 185
 Tang::ComputedExpressionCompiledFunction, 198

Tang::ComputedExpressionError, 211
Tang::ComputedExpressionFloat, 224
Tang::ComputedExpressionInteger, 238
Tang::ComputedExpressionIterator, 252
Tang::ComputedExpressionIteratorEnd, 266
Tang::ComputedExpressionLibrary, 280
Tang::ComputedExpressionLibraryMath, 294
Tang::ComputedExpressionMap, 309
Tang::ComputedExpressionNativeBoundFunction, 322
Tang::ComputedExpressionNativeFunction, 338
Tang::ComputedExpressionNativeLibraryFunction, 352
Tang::ComputedExpressionString, 369

__modulo
Tang::ComputedExpression, 157
Tang::ComputedExpressionArray, 171
Tang::ComputedExpressionBoolean, 185
Tang::ComputedExpressionCompiledFunction, 198
Tang::ComputedExpressionError, 211
Tang::ComputedExpressionFloat, 224
Tang::ComputedExpressionInteger, 238
Tang::ComputedExpressionIterator, 253
Tang::ComputedExpressionIteratorEnd, 267
Tang::ComputedExpressionLibrary, 281
Tang::ComputedExpressionLibraryMath, 295
Tang::ComputedExpressionMap, 308
Tang::ComputedExpressionNativeBoundFunction, 324
Tang::ComputedExpressionNativeFunction, 338
Tang::ComputedExpressionNativeLibraryFunction, 354
Tang::ComputedExpressionString, 370

__multiply
Tang::ComputedExpression, 157
Tang::ComputedExpressionArray, 171
Tang::ComputedExpressionBoolean, 185
Tang::ComputedExpressionCompiledFunction, 198
Tang::ComputedExpressionError, 211
Tang::ComputedExpressionFloat, 225
Tang::ComputedExpressionInteger, 239
Tang::ComputedExpressionIterator, 253
Tang::ComputedExpressionIteratorEnd, 267
Tang::ComputedExpressionLibrary, 281
Tang::ComputedExpressionLibraryMath, 295
Tang::ComputedExpressionMap, 309
Tang::ComputedExpressionNativeBoundFunction, 324
Tang::ComputedExpressionNativeFunction, 338
Tang::ComputedExpressionNativeLibraryFunction, 354
Tang::ComputedExpressionString, 370

__negative
Tang::ComputedExpression, 158
Tang::ComputedExpressionArray, 172
Tang::ComputedExpressionBoolean, 186
Tang::ComputedExpressionCompiledFunction, 199
Tang::ComputedExpressionError, 212

Tang::ComputedExpressionFloat, 225
Tang::ComputedExpressionInteger, 240
Tang::ComputedExpressionIterator, 254
Tang::ComputedExpressionIteratorEnd, 268
Tang::ComputedExpressionLibrary, 281
Tang::ComputedExpressionLibraryMath, 295
Tang::ComputedExpressionMap, 309
Tang::ComputedExpressionNativeBoundFunction, 324
Tang::ComputedExpressionNativeFunction, 339
Tang::ComputedExpressionNativeLibraryFunction, 354
Tang::ComputedExpressionString, 371

__not
Tang::ComputedExpression, 158
Tang::ComputedExpressionArray, 172
Tang::ComputedExpressionBoolean, 186
Tang::ComputedExpressionCompiledFunction, 199
Tang::ComputedExpressionError, 212
Tang::ComputedExpressionFloat, 225
Tang::ComputedExpressionInteger, 240
Tang::ComputedExpressionIterator, 254
Tang::ComputedExpressionIteratorEnd, 268
Tang::ComputedExpressionLibrary, 282
Tang::ComputedExpressionLibraryMath, 296
Tang::ComputedExpressionMap, 309
Tang::ComputedExpressionNativeBoundFunction, 325
Tang::ComputedExpressionNativeFunction, 339
Tang::ComputedExpressionNativeLibraryFunction, 355
Tang::ComputedExpressionString, 371

__period
Tang::ComputedExpression, 158
Tang::ComputedExpressionArray, 172
Tang::ComputedExpressionBoolean, 186
Tang::ComputedExpressionCompiledFunction, 199
Tang::ComputedExpressionError, 212
Tang::ComputedExpressionFloat, 226
Tang::ComputedExpressionInteger, 240
Tang::ComputedExpressionIterator, 254
Tang::ComputedExpressionIteratorEnd, 268
Tang::ComputedExpressionLibrary, 282
Tang::ComputedExpressionLibraryMath, 296
Tang::ComputedExpressionMap, 309
Tang::ComputedExpressionNativeBoundFunction, 325
Tang::ComputedExpressionNativeFunction, 339
Tang::ComputedExpressionNativeLibraryFunction, 355
Tang::ComputedExpressionString, 371

__slice
Tang::ComputedExpression, 159
Tang::ComputedExpressionArray, 173
Tang::ComputedExpressionBoolean, 187
Tang::ComputedExpressionCompiledFunction, 200
Tang::ComputedExpressionError, 213
Tang::ComputedExpressionFloat, 226

Tang::ComputedExpressionInteger, 240
 Tang::ComputedExpressionIterator, 255
 Tang::ComputedExpressionIteratorEnd, 268
 Tang::ComputedExpressionLibrary, 282
 Tang::ComputedExpressionLibraryMath, 296
 Tang::ComputedExpressionMap, 310
 Tang::ComputedExpressionNativeBoundFunction, 325
 Tang::ComputedExpressionNativeFunction, 340
 Tang::ComputedExpressionNativeLibraryFunction, 355
 Tang::ComputedExpressionString, 372
string
 Tang::ComputedExpression, 159
 Tang::ComputedExpressionArray, 173
 Tang::ComputedExpressionBoolean, 187
 Tang::ComputedExpressionCompiledFunction, 200
 Tang::ComputedExpressionError, 213
 Tang::ComputedExpressionFloat, 227
 Tang::ComputedExpressionInteger, 241
 Tang::ComputedExpressionIterator, 255
 Tang::ComputedExpressionIteratorEnd, 269
 Tang::ComputedExpressionLibrary, 283
 Tang::ComputedExpressionLibraryMath, 297
 Tang::ComputedExpressionMap, 310
 Tang::ComputedExpressionNativeBoundFunction, 326
 Tang::ComputedExpressionNativeFunction, 340
 Tang::ComputedExpressionNativeLibraryFunction, 356
 Tang::ComputedExpressionString, 373
subtract
 Tang::ComputedExpression, 159
 Tang::ComputedExpressionArray, 174
 Tang::ComputedExpressionBoolean, 187
 Tang::ComputedExpressionCompiledFunction, 200
 Tang::ComputedExpressionError, 213
 Tang::ComputedExpressionFloat, 227
 Tang::ComputedExpressionInteger, 241
 Tang::ComputedExpressionIterator, 255
 Tang::ComputedExpressionIteratorEnd, 269
 Tang::ComputedExpressionLibrary, 283
 Tang::ComputedExpressionLibraryMath, 297
 Tang::ComputedExpressionMap, 311
 Tang::ComputedExpressionNativeBoundFunction, 326
 Tang::ComputedExpressionNativeFunction, 340
 Tang::ComputedExpressionNativeLibraryFunction, 356
 Tang::ComputedExpressionString, 373
~GarbageCollected
 Tang::GarbageCollected, 385
 ADD
 opcode.hpp, 489
 Add
 Tang::AstNodeBinary, 32
 addBreak
 Tang::Program, 408
 addBytecode
 Tang::Program, 409
 addContinue
 Tang::Program, 409
 addIdentifier
 Tang::Program, 409
 addIdentifierAssigned
 Tang::Program, 410
 addLibraryAlias
 Tang::Program, 410
 addString
 Tang::Program, 410
 And
 Tang::AstNodeBinary, 32
 append
 Tang::ComputedExpressionArray, 174
 ARRAY
 opcode.hpp, 489
 ASSIGNINDEX
 opcode.hpp, 489
 AstNode
 Tang::AstNode, 18
 AstNodeArray
 Tang::AstNodeArray, 23
 AstNodeAssign
 Tang::AstNodeAssign, 27
 AstNodeBinary
 Tang::AstNodeBinary, 33
 AstNodeBlock
 Tang::AstNodeBlock, 37
 AstNodeBoolean
 Tang::AstNodeBoolean, 41
 AstNodeBreak
 Tang::AstNodeBreak, 45
 AstNodeCast
 Tang::AstNodeCast, 50
 AstNodeContinue
 Tang::AstNodeContinue, 54
 AstNodeDoWhile
 Tang::AstNodeDoWhile, 58
 AstNodeFloat
 Tang::AstNodeFloat, 62
 AstNodeFor
 Tang::AstNodeFor, 67
 AstNodeFunctionCall
 Tang::AstNodeFunctionCall, 71
 AstNodeFunctionDeclaration
 Tang::AstNodeFunctionDeclaration, 74
 AstNodeIdentifier
 Tang::AstNodeIdentifier, 79
 AstNodeIfElse
 Tang::AstNodeIfElse, 84
 AstNodeIndex
 Tang::AstNodeIndex, 88
 AstNodeInteger
 Tang::AstNodeInteger, 93
 AstNodeLibrary
 Tang::AstNodeLibrary, 97

AstNodeMap
 Tang::AstNodeMap, 101

AstNodePeriod
 Tang::AstNodePeriod, 105

AstNodePrint
 Tang::AstNodePrint, 110

AstNodeRangedFor
 Tang::AstNodeRangedFor, 114

AstNodeReturn
 Tang::AstNodeReturn, 118

AstNodeSlice
 Tang::AstNodeSlice, 123

AstNodeString
 Tang::AstNodeString, 127, 128

AstNodeTernary
 Tang::AstNodeTernary, 133

AstNodeUnary
 Tang::AstNodeUnary, 138

AstNodeUse
 Tang::AstNodeUse, 143

AstNodeWhile
 Tang::AstNodeWhile, 148

BOOLEAN
 opcode.hpp, 489

Boolean
 Tang::AstNodeCast, 50

build/generated/location.hh, 435

bytesLength
 Tang::ComputedExpressionString, 373
 Tang::UnicodeString, 429

CALLFUNC
 opcode.hpp, 489

CASTBOOLEAN
 opcode.hpp, 489

CASTFLOAT
 opcode.hpp, 489

CASTINTEGER
 opcode.hpp, 489

CASTSTRING
 opcode.hpp, 489

CodeType
 Tang::Program, 408

compile
 Tang::AstNode, 19
 Tang::AstNodeArray, 24
 Tang::AstNodeAssign, 28
 Tang::AstNodeBinary, 33
 Tang::AstNodeBlock, 38
 Tang::AstNodeBoolean, 42
 Tang::AstNodeBreak, 45
 Tang::AstNodeCast, 50
 Tang::AstNodeContinue, 54
 Tang::AstNodeDoWhile, 59
 Tang::AstNodeFloat, 63
 Tang::AstNodeFor, 67
 Tang::AstNodeFunctionCall, 71
 Tang::AstNodeFunctionDeclaration, 75

 Tang::AstNodeIdentifier, 79
 Tang::AstNodeIfElse, 84
 Tang::AstNodeIndex, 89
 Tang::AstNodeInteger, 94
 Tang::AstNodeLibrary, 98
 Tang::AstNodeMap, 101
 Tang::AstNodePeriod, 106
 Tang::AstNodePrint, 110
 Tang::AstNodeRangedFor, 114
 Tang::AstNodeReturn, 119
 Tang::AstNodeSlice, 123
 Tang::AstNodeString, 128
 Tang::AstNodeTernary, 134
 Tang::AstNodeUnary, 138
 Tang::AstNodeUse, 144
 Tang::AstNodeWhile, 149

compileLiteral
 Tang::AstNodeString, 129

compilePreprocess
 Tang::AstNode, 19
 Tang::AstNodeArray, 24
 Tang::AstNodeAssign, 28
 Tang::AstNodeBinary, 34
 Tang::AstNodeBlock, 38
 Tang::AstNodeBoolean, 42
 Tang::AstNodeBreak, 46
 Tang::AstNodeCast, 51
 Tang::AstNodeContinue, 55
 Tang::AstNodeDoWhile, 59
 Tang::AstNodeFloat, 63
 Tang::AstNodeFor, 68
 Tang::AstNodeFunctionCall, 71
 Tang::AstNodeFunctionDeclaration, 75
 Tang::AstNodeIdentifier, 79
 Tang::AstNodeIfElse, 85
 Tang::AstNodeIndex, 89
 Tang::AstNodeInteger, 94
 Tang::AstNodeLibrary, 98
 Tang::AstNodeMap, 102
 Tang::AstNodePeriod, 106
 Tang::AstNodePrint, 111
 Tang::AstNodeRangedFor, 115
 Tang::AstNodeReturn, 119
 Tang::AstNodeSlice, 124
 Tang::AstNodeString, 129
 Tang::AstNodeTernary, 134
 Tang::AstNodeUnary, 140
 Tang::AstNodeUse, 144
 Tang::AstNodeWhile, 149

compileScript
 Tang::TangBase, 422

compileTemplate
 Tang::TangBase, 422

ComputedExpressionArray
 Tang::ComputedExpressionArray, 166

ComputedExpressionBoolean
 Tang::ComputedExpressionBoolean, 181

ComputedExpressionCompiledFunction

Tang::ComputedExpressionCompiledFunction, 193
 ComputedExpressionError
 Tang::ComputedExpressionError, 207
 ComputedExpressionFloat
 Tang::ComputedExpressionFloat, 219
 ComputedExpressionInteger
 Tang::ComputedExpressionInteger, 233
 ComputedExpressionIterator
 Tang::ComputedExpressionIterator, 248
 ComputedExpressionMap
 Tang::ComputedExpressionMap, 304
 ComputedExpressionNativeBoundFunction
 Tang::ComputedExpressionNativeBoundFunction, 317
 ComputedExpressionNativeFunction
 Tang::ComputedExpressionNativeFunction, 333
 ComputedExpressionNativeLibraryFunction
 Tang::ComputedExpressionNativeLibraryFunction, 348
 ComputedExpressionString
 Tang::ComputedExpressionString, 363
 COPY
 opcode.hpp, 488
 currentIndex
 Tang::SingletonObjectPool< T >, 419
 currentRecycledIndex
 Tang::SingletonObjectPool< T >, 419

 Default
 Tang::AstNode, 18
 Tang::AstNodeArray, 23
 Tang::AstNodeAssign, 27
 Tang::AstNodeBinary, 33
 Tang::AstNodeBlock, 37
 Tang::AstNodeBoolean, 41
 Tang::AstNodeBreak, 45
 Tang::AstNodeCast, 49
 Tang::AstNodeContinue, 54
 Tang::AstNodeDoWhile, 58
 Tang::AstNodeFloat, 62
 Tang::AstNodeFor, 66
 Tang::AstNodeFunctionCall, 70
 Tang::AstNodeFunctionDeclaration, 74
 Tang::AstNodeIdentifier, 78
 Tang::AstNodeIfElse, 83
 Tang::AstNodeIndex, 88
 Tang::AstNodeInteger, 93
 Tang::AstNodeLibrary, 97
 Tang::AstNodeMap, 101
 Tang::AstNodePeriod, 105
 Tang::AstNodePrint, 110
 Tang::AstNodeRangedFor, 113
 Tang::AstNodeReturn, 118
 Tang::AstNodeSlice, 122
 Tang::AstNodeString, 127
 Tang::AstNodeTernary, 133
 Tang::AstNodeUnary, 138
 Tang::AstNodeUse, 143
 Tang::AstNodeWhile, 148

 DIVIDE
 opcode.hpp, 489
 Divide
 Tang::AstNodeBinary, 32
 dump
 Tang::AstNode, 20
 Tang::AstNodeArray, 25
 Tang::AstNodeAssign, 29
 Tang::AstNodeBinary, 34
 Tang::AstNodeBlock, 39
 Tang::AstNodeBoolean, 43
 Tang::AstNodeBreak, 47
 Tang::AstNodeCast, 51
 Tang::AstNodeContinue, 55
 Tang::AstNodeDoWhile, 60
 Tang::AstNodeFloat, 64
 Tang::AstNodeFor, 68
 Tang::AstNodeFunctionCall, 72
 Tang::AstNodeFunctionDeclaration, 76
 Tang::AstNodeIdentifier, 80
 Tang::AstNodeIfElse, 85
 Tang::AstNodeIndex, 90
 Tang::AstNodeInteger, 95
 Tang::AstNodeLibrary, 99
 Tang::AstNodeMap, 103
 Tang::AstNodePeriod, 107
 Tang::AstNodePrint, 111
 Tang::AstNodeRangedFor, 116
 Tang::AstNodeReturn, 120
 Tang::AstNodeSlice, 124
 Tang::AstNodeString, 130
 Tang::AstNodeTernary, 135
 Tang::AstNodeUnary, 140
 Tang::AstNodeUse, 145
 Tang::AstNodeWhile, 150
 Tang::ComputedExpression, 160
 Tang::ComputedExpressionArray, 175
 Tang::ComputedExpressionBoolean, 188
 Tang::ComputedExpressionCompiledFunction, 201
 Tang::ComputedExpressionError, 214
 Tang::ComputedExpressionFloat, 228
 Tang::ComputedExpressionInteger, 242
 Tang::ComputedExpressionIterator, 256
 Tang::ComputedExpressionIteratorEnd, 270
 Tang::ComputedExpressionLibrary, 284
 Tang::ComputedExpressionLibraryMath, 298
 Tang::ComputedExpressionMap, 311
 Tang::ComputedExpressionNativeBoundFunction, 327
 Tang::ComputedExpressionNativeFunction, 341
 Tang::ComputedExpressionNativeLibraryFunction, 357
 Tang::ComputedExpressionString, 373
 dumpBytecode
 Tang::Program, 411
 DUMPPROGRAMCHECK
 program-dumpBytecode.cpp, 528

 EQ

opcode.hpp, 489
Equal
 Tang::AstNodeBinary, 32
Error
 Tang::Error, 381
error.cpp
 operator<<, 526
execute
 Tang::Program, 411
EXECUTEPROGRAMCHECK
 program-execute.cpp, 530

FLOAT
 opcode.hpp, 489
Float
 Tang::AstNodeCast, 50
FUNCTION
 opcode.hpp, 489
functionsDeclared
 Tang::Program, 416

GarbageCollected
 Tang::GarbageCollected, 384, 385
garbageCollected.cpp
 operator<<, 527
get
 Tang::SingletonObjectPool< T >, 418
get_next_token
 Tang::HtmlEscape, 400
 Tang::HtmlEscapeAscii, 402
 Tang::TangScanner, 425
 Tang::Unescape, 426
getArgc
 Tang::ComputedExpressionNativeBoundFunction, 327
 Tang::ComputedExpressionNativeFunction, 341
getAst
 Tang::Program, 412
getBytecode
 Tang::Program, 412
getCode
 Tang::Program, 412
getCollection
 Tang::AstNodeIndex, 90
getContents
 Tang::ComputedExpressionArray, 175
getFunction
 Tang::ComputedExpressionNativeBoundFunction, 327
 Tang::ComputedExpressionNativeFunction, 341
 Tang::ComputedExpressionNativeLibraryFunction, 357
getIdentifiers
 Tang::Program, 413
getIdentifiersAssigned
 Tang::Program, 413
getIndex
 Tang::AstNodeIndex, 90
getInstance

 Tang::SingletonObjectPool< T >, 418
GETITERATOR
 opcode.hpp, 489
getLibraryAliases
 Tang::Program, 413
getLibraryAttributes
 Tang::ComputedExpressionLibraryMath, 298
getMethods
 Tang::ComputedExpressionArray, 175
 Tang::ComputedExpressionString, 374
getResult
 Tang::Program, 413
getStrings
 Tang::Program, 414
getTargetTypeIndex
 Tang::ComputedExpressionNativeBoundFunction, 327
getValue
 Tang::ComputedExpressionFloat, 228
 Tang::ComputedExpressionInteger, 242
 Tang::ComputedExpressionString, 374
GreaterThan
 Tang::AstNodeBinary, 32
GreaterThanOrEqualTo
 Tang::AstNodeBinary, 32
GT
 opcode.hpp, 489
GTE
 opcode.hpp, 489

HtmlEscape
 Tang::HtmlEscape, 399
htmlEscape
 unicodeString.hpp, 497
HtmlEscapeAscii
 Tang::HtmlEscapeAscii, 401
htmlEscapeAscii
 unicodeString.hpp, 497

include/astNode.hpp, 437
include/astNodeArray.hpp, 438
include/astNodeAssign.hpp, 439
include/astNodeBinary.hpp, 440
include/astNodeBlock.hpp, 441
include/astNodeBoolean.hpp, 442
include/astNodeBreak.hpp, 443
include/astNodeCast.hpp, 444
include/astNodeContinue.hpp, 445
include/astNodeDoWhile.hpp, 446
include/astNodeFloat.hpp, 447
include/astNodeFor.hpp, 448
include/astNodeFunctionCall.hpp, 449
include/astNodeFunctionDeclaration.hpp, 450
include/astNodeIdentifier.hpp, 451
include/astNodeElse.hpp, 452
include/astNodeIndex.hpp, 453
include/astNodeInteger.hpp, 454
include/astNodeLibrary.hpp, 455
include/astNodeMap.hpp, 456

include/astNodePeriod.hpp, 457
 include/astNodePrint.hpp, 458
 include/astNodeRangedFor.hpp, 459
 include/astNodeReturn.hpp, 460
 include/astNodeSlice.hpp, 461
 include/astNodeString.hpp, 462
 include/astNodeTernary.hpp, 462
 include/astNodeUnary.hpp, 463
 include/astNodeUse.hpp, 464
 include/astNodeWhile.hpp, 465
 include/computedExpression.hpp, 466
 include/computedExpressionArray.hpp, 467
 include/computedExpressionBoolean.hpp, 468
 include/computedExpressionCompiledFunction.hpp,
 469
 include/computedExpressionError.hpp, 470
 include/computedExpressionFloat.hpp, 471
 include/computedExpressionInteger.hpp, 472
 include/computedExpressionIterator.hpp, 472
 include/computedExpressionIteratorEnd.hpp, 473
 include/computedExpressionLibrary.hpp, 474
 include/computedExpressionLibraryMath.hpp, 475
 include/computedExpressionMap.hpp, 477
 include/computedExpressionNativeBoundFunction.hpp,
 478
 include/computedExpressionNativeFunction.hpp, 479
 include/computedExpressionNativeLibraryFunction.hpp,
 480
 include/computedExpressionString.hpp, 481
 include/context.hpp, 481
 include/error.hpp, 482
 include/garbageCollected.hpp, 483
 include/htmlEscape.hpp, 484
 include/htmlEscapeAscii.hpp, 485
 include/macros.hpp, 487
 include/opcode.hpp, 488
 include/program.hpp, 490
 include/singletonObjectPool.hpp, 491
 include/tang.hpp, 492
 include/tangBase.hpp, 493
 include/tangScanner.hpp, 494
 include/unescape.hpp, 495
 include/unicodeString.hpp, 496
 INDEX
 opcode.hpp, 489
 INTEGER
 opcode.hpp, 489
 Integer
 Tang::AstNodeCast, 50
 is_equal
 Tang::ComputedExpression, 160–162
 Tang::ComputedExpressionArray, 175–177
 Tang::ComputedExpressionBoolean, 188–190
 Tang::ComputedExpressionCompiledFunction,
 201–203
 Tang::ComputedExpressionError, 214–216
 Tang::ComputedExpressionFloat, 228–230
 Tang::ComputedExpressionInteger, 242–244
 Tang::ComputedExpressionIterator, 256, 258, 259
 Tang::ComputedExpressionIteratorEnd, 270, 272,
 273
 Tang::ComputedExpressionLibrary, 284, 286, 287
 Tang::ComputedExpressionLibraryMath, 298–300
 Tang::ComputedExpressionMap, 311–313
 Tang::ComputedExpressionNativeBoundFunction,
 328–330
 Tang::ComputedExpressionNativeFunction, 342–
 344
 Tang::ComputedExpressionNativeLibraryFunction,
 357–359
 Tang::ComputedExpressionString, 374–377
 IsAssignment
 Tang::AstNode, 18
 Tang::AstNodeArray, 23
 Tang::AstNodeAssign, 27
 Tang::AstNodeBinary, 33
 Tang::AstNodeBlock, 37
 Tang::AstNodeBoolean, 41
 Tang::AstNodeBreak, 45
 Tang::AstNodeCast, 49
 Tang::AstNodeContinue, 54
 Tang::AstNodeDoWhile, 58
 Tang::AstNodeFloat, 62
 Tang::AstNodeFor, 66
 Tang::AstNodeFunctionCall, 70
 Tang::AstNodeFunctionDeclaration, 74
 Tang::AstNodeIdentifier, 78
 Tang::AstNodeIfElse, 83
 Tang::AstNodeIndex, 88
 Tang::AstNodeInteger, 93
 Tang::AstNodeLibrary, 97
 Tang::AstNodeMap, 101
 Tang::AstNodePeriod, 105
 Tang::AstNodePrint, 110
 Tang::AstNodeRangedFor, 113
 Tang::AstNodeReturn, 118
 Tang::AstNodeSlice, 122
 Tang::AstNodeString, 127
 Tang::AstNodeTernary, 133
 Tang::AstNodeUnary, 138
 Tang::AstNodeUse, 143
 Tang::AstNodeWhile, 148
 isCopyNeeded
 Tang::ComputedExpression, 162
 Tang::ComputedExpressionArray, 178
 Tang::ComputedExpressionBoolean, 190
 Tang::ComputedExpressionCompiledFunction, 203
 Tang::ComputedExpressionError, 216
 Tang::ComputedExpressionFloat, 230
 Tang::ComputedExpressionInteger, 245
 Tang::ComputedExpressionIterator, 259
 Tang::ComputedExpressionIteratorEnd, 273
 Tang::ComputedExpressionLibrary, 287
 Tang::ComputedExpressionLibraryMath, 300
 Tang::ComputedExpressionMap, 314

Tang::ComputedExpressionNativeBoundFunction, 330
Tang::ComputedExpressionNativeFunction, 345
Tang::ComputedExpressionNativeLibraryFunction, 359
Tang::ComputedExpressionString, 377
Tang::GarbageCollected, 385
ISITERATOREND
 opcode.hpp, 489
ITERATORNEXT
 opcode.hpp, 489
JMP
 opcode.hpp, 489
JMPF
 opcode.hpp, 489
JMPF_POP
 opcode.hpp, 489
JMPT
 opcode.hpp, 489
JMPT_POP
 opcode.hpp, 489
length
 Tang::ComputedExpressionString, 377
 Tang::UnicodeString, 429
LessThan
 Tang::AstNodeBinary, 32
LessThanEqual
 Tang::AstNodeBinary, 32
LIBRARY
 opcode.hpp, 489
LIBRARYCOPY
 opcode.hpp, 489
LIBRARYSAVE
 opcode.hpp, 489
location.hh
 operator<<, 436, 437
LT
 opcode.hpp, 489
LTE
 opcode.hpp, 489
make
 Tang::GarbageCollected, 386
make_shared
 Tang::TangBase, 422
makeCopy
 Tang::ComputedExpression, 163
 Tang::ComputedExpressionArray, 178
 Tang::ComputedExpressionBoolean, 191
 Tang::ComputedExpressionCompiledFunction, 204
 Tang::ComputedExpressionError, 217
 Tang::ComputedExpressionFloat, 231
 Tang::ComputedExpressionInteger, 245
 Tang::ComputedExpressionIterator, 259
 Tang::ComputedExpressionIteratorEnd, 273
 Tang::ComputedExpressionLibrary, 287
 Tang::ComputedExpressionLibraryMath, 301
Tang::ComputedExpressionMap, 314
Tang::ComputedExpressionNativeBoundFunction, 330
Tang::ComputedExpressionNativeFunction, 345
Tang::ComputedExpressionNativeLibraryFunction, 360
Tang::ComputedExpressionString, 378
Tang::GarbageCollected, 386
MAP
 opcode.hpp, 489
MODULO
 opcode.hpp, 489
Modulo
 Tang::AstNodeBinary, 32
MULTIPLY
 opcode.hpp, 489
Multiply
 Tang::AstNodeBinary, 32
NEGATIVE
 opcode.hpp, 489
Negative
 Tang::AstNodeUnary, 138
NEQ
 opcode.hpp, 489
NOT
 opcode.hpp, 489
Not
 Tang::AstNodeUnary, 138
NotEqual
 Tang::AstNodeBinary, 32
NULLVAL
 opcode.hpp, 489
Opcode
 opcode.hpp, 488
opcode.hpp
 ADD, 489
 ARRAY, 489
 ASSIGNINDEX, 489
 BOOLEAN, 489
 CALLFUNC, 489
 CASTBOOLEAN, 489
 CASTFLOAT, 489
 CASTINTEGER, 489
 CASTSTRING, 489
 COPY, 488
 DIVIDE, 489
 EQ, 489
 FLOAT, 489
 FUNCTION, 489
 GETITERATOR, 489
 GT, 489
 GTE, 489
 INDEX, 489
 INTEGER, 489
 ISITERATOREND, 489
 ITERATORNEXT, 489
 JMP, 489

JMPF, 489
 JMPF_POP, 489
 JMPT, 489
 JMPT_POP, 489
 LIBRARY, 489
 LIBRARYCOPY, 489
 LIBRARYSAVE, 489
 LT, 489
 LTE, 489
 MAP, 489
 MODULO, 489
 MULTIPLY, 489
 NEGATIVE, 489
 NEQ, 489
 NOT, 489
 NULLVAL, 489
 Opcode, 488
 PEEK, 488
 PERIOD, 489
 POKE, 488
 POP, 488
 PRINT, 489
 RETURN, 489
 SLICE, 489
 STRING, 489
 SUBTRACT, 489
 Operation
 Tang::AstNodeBinary, 32
 Operator
 Tang::AstNodeUnary, 137
 operator std::string
 Tang::UnicodeString, 429
 operator!
 Tang::GarbageCollected, 387
 operator!=
 Tang::GarbageCollected, 387
 operator<
 Tang::GarbageCollected, 392
 Tang::UnicodeString, 430
 operator<<
 error.cpp, 526
 garbageCollected.cpp, 527
 location.hh, 436, 437
 Tang::Error, 381
 Tang::GarbageCollected, 397
 operator<=
 Tang::GarbageCollected, 392
 operator>
 Tang::GarbageCollected, 396
 operator>=
 Tang::GarbageCollected, 397
 operator*
 Tang::GarbageCollected, 388, 389
 operator+
 Tang::GarbageCollected, 389
 Tang::UnicodeString, 430
 operator+=
 Tang::ComputedExpressionString, 378
 Tang::UnicodeString, 430
 operator-
 Tang::GarbageCollected, 390
 operator->
 Tang::GarbageCollected, 391
 operator/
 Tang::GarbageCollected, 391
 operator=
 Tang::GarbageCollected, 393
 operator==
 Tang::GarbageCollected, 393–396
 Tang::UnicodeString, 431
 operator%
 Tang::GarbageCollected, 388
 Or
 Tang::AstNodeBinary, 32
 PEEK
 opcode.hpp, 488
 PERIOD
 opcode.hpp, 489
 POKE
 opcode.hpp, 488
 POP
 opcode.hpp, 488
 popBreakStack
 Tang::Program, 414
 popContinueStack
 Tang::Program, 414
 PreprocessState
 Tang::AstNode, 18
 Tang::AstNodeArray, 23
 Tang::AstNodeAssign, 27
 Tang::AstNodeBinary, 33
 Tang::AstNodeBlock, 37
 Tang::AstNodeBoolean, 41
 Tang::AstNodeBreak, 45
 Tang::AstNodeCast, 49
 Tang::AstNodeContinue, 54
 Tang::AstNodeDoWhile, 58
 Tang::AstNodeFloat, 62
 Tang::AstNodeFor, 66
 Tang::AstNodeFunctionCall, 70
 Tang::AstNodeFunctionDeclaration, 74
 Tang::AstNodeIdentifier, 78
 Tang::AstNodeIfElse, 83
 Tang::AstNodeIndex, 88
 Tang::AstNodeInteger, 93
 Tang::AstNodeLibrary, 97
 Tang::AstNodeMap, 101
 Tang::AstNodePeriod, 105
 Tang::AstNodePrint, 109
 Tang::AstNodeRangedFor, 113
 Tang::AstNodeReturn, 118
 Tang::AstNodeSlice, 122
 Tang::AstNodeString, 127
 Tang::AstNodeTernary, 133
 Tang::AstNodeUnary, 138
 Tang::AstNodeUse, 143

Tang::AstNodeWhile, 148
PRINT
 opcode.hpp, 489
Program
 Tang::Program, 408
program-dumpBytecode.cpp
 DUMPPROGRAMCHECK, 528
program-execute.cpp
 EXECUTEPROGRAMCHECK, 530
 STACKCHECK, 530
pushEnvironment
 Tang::Program, 415

recycle
 Tang::SingletonObjectPool< T >, 419
render
 Tang::UnicodeString, 431
renderAscii
 Tang::UnicodeString, 432
RETURN
 opcode.hpp, 489

Script
 Tang::Program, 408
setFunctionStackDeclaration
 Tang::Program, 415
setJumpTarget
 Tang::Program, 416
SLICE
 opcode.hpp, 489
src/astNode.cpp, 499
src/astNodeArray.cpp, 499
src/astNodeAssign.cpp, 500
src/astNodeBinary.cpp, 500
src/astNodeBlock.cpp, 501
src/astNodeBoolean.cpp, 502
src/astNodeBreak.cpp, 502
src/astNodeCast.cpp, 503
src/astNodeContinue.cpp, 503
src/astNodeDoWhile.cpp, 504
src/astNodeFloat.cpp, 504
src/astNodeFor.cpp, 505
src/astNodeFunctionCall.cpp, 505
src/astNodeFunctionDeclaration.cpp, 506
src/astNodeIdentifier.cpp, 506
src/astNodeIfElse.cpp, 507
src/astNodeIndex.cpp, 508
src/astNodeInteger.cpp, 508
src/astNodeLibrary.cpp, 509
src/astNodeMap.cpp, 509
src/astNodePeriod.cpp, 510
src/astNodePrint.cpp, 511
src/astNodeRangedFor.cpp, 511
src/astNodeReturn.cpp, 512
src/astNodeSlice.cpp, 512
src/astNodeString.cpp, 513
src/astNodeTernary.cpp, 513
src/astNodeUnary.cpp, 514
src/astNodeUse.cpp, 514

 src/astNodeWhile.cpp, 515
 src/computedExpression.cpp, 515
 src/computedExpressionArray.cpp, 516
 src/computedExpressionBoolean.cpp, 517
 src/computedExpressionCompiledFunction.cpp, 517
 src/computedExpressionError.cpp, 518
 src/computedExpressionFloat.cpp, 518
 src/computedExpressionInteger.cpp, 519
 src/computedExpressionIterator.cpp, 520
 src/computedExpressionIteratorEnd.cpp, 520
 src/computedExpressionLibrary.cpp, 521
 src/computedExpressionLibraryMath.cpp, 521
 src/computedExpressionMap.cpp, 522
 src/computedExpressionNativeBoundFunction.cpp, 523
 src/computedExpressionNativeFunction.cpp, 523
 src/computedExpressionNativeLibraryFunction.cpp, 524
 src/computedExpressionString.cpp, 524
 src/context.cpp, 525
 src/error.cpp, 526
 src/garbageCollected.cpp, 527
 src/program-dumpBytecode.cpp, 528
 src/program-execute.cpp, 529
 src/program.cpp, 531
 src/tangBase.cpp, 531
 src/unicodeString.cpp, 532
 STACKCHECK
 program-execute.cpp, 530
STRING
 opcode.hpp, 489
String
 Tang::AstNodeCast, 50
substr
 Tang::UnicodeString, 432
SUBTRACT
 opcode.hpp, 489
Subtract
 Tang::AstNodeBinary, 32

Tang::AstNode, 15
 AstNode, 18
 compile, 19
 compilePreprocess, 19
 Default, 18
 dump, 20
 IsAssignment, 18
 PreprocessState, 18
Tang::AstNodeArray, 21
 AstNodeArray, 23
 compile, 24
 compilePreprocess, 24
 Default, 23
 dump, 25
 IsAssignment, 23
 PreprocessState, 23
Tang::AstNodeAssign, 25
 AstNodeAssign, 27
 compile, 28
 compilePreprocess, 28
 Default, 27

dump, 29
IsAssignment, 27
PreprocessState, 27
Tang::AstNodeBinary, 29
Add, 32
And, 32
AstNodeBinary, 33
compile, 33
compilePreprocess, 34
Default, 33
Divide, 32
dump, 34
Equal, 32
GreaterThan, 32
GreaterThanOrEqualTo, 32
IsAssignment, 33
LessThan, 32
LessThanOrEqualTo, 32
Modulo, 32
Multiply, 32
NotEqual, 32
Operation, 32
Or, 32
PreprocessState, 33
Subtract, 32
Tang::AstNodeBlock, 35
AstNodeBlock, 37
compile, 38
compilePreprocess, 38
Default, 37
dump, 39
IsAssignment, 37
PreprocessState, 37
Tang::AstNodeBoolean, 39
AstNodeBoolean, 41
compile, 42
compilePreprocess, 42
Default, 41
dump, 43
IsAssignment, 41
PreprocessState, 41
Tang::AstNodeBreak, 43
AstNodeBreak, 45
compile, 45
compilePreprocess, 46
Default, 45
dump, 47
IsAssignment, 45
PreprocessState, 45
Tang::AstNodeCast, 47
AstNodeCast, 50
Boolean, 50
compile, 50
compilePreprocess, 51
Default, 49
dump, 51
Float, 50
Integer, 50
IsAssignment, 49
PreprocessState, 49
String, 50
Type, 49
Tang::AstNodeContinue, 52
AstNodeContinue, 54
compile, 54
compilePreprocess, 55
Default, 54
dump, 55
IsAssignment, 54
PreprocessState, 54
Tang::AstNodeDoWhile, 56
AstNodeDoWhile, 58
compile, 59
compilePreprocess, 59
Default, 58
dump, 60
IsAssignment, 58
PreprocessState, 58
Tang::AstNodeFloat, 60
AstNodeFloat, 62
compile, 63
compilePreprocess, 63
Default, 62
dump, 64
IsAssignment, 62
PreprocessState, 62
Tang::AstNodeFor, 64
AstNodeFor, 67
compile, 67
compilePreprocess, 68
Default, 66
dump, 68
IsAssignment, 66
PreprocessState, 66
Tang::AstNodeFunctionCall, 69
AstNodeFunctionCall, 71
compile, 71
compilePreprocess, 71
Default, 70
dump, 72
IsAssignment, 70
PreprocessState, 70
Tang::AstNodeFunctionDeclaration, 72
AstNodeFunctionDeclaration, 74
compile, 75
compilePreprocess, 75
Default, 74
dump, 76
IsAssignment, 74
PreprocessState, 74
Tang::AstNodeIdentifier, 76
AstNodeIdentifier, 79
compile, 79
compilePreprocess, 79
Default, 78
dump, 80

IsAssignment, 78
PreprocessState, 78
Tang::AstNodeIfElse, 81
AstNodeIfElse, 84
compile, 84
compilePreprocess, 85
Default, 83
dump, 85
IsAssignment, 83
PreprocessState, 83
Tang::AstNodeIndex, 86
AstNodeIndex, 88
compile, 89
compilePreprocess, 89
Default, 88
dump, 90
getCollection, 90
getIndex, 90
IsAssignment, 88
PreprocessState, 88
Tang::AstNodeInteger, 91
AstNodeInteger, 93
compile, 94
compilePreprocess, 94
Default, 93
dump, 95
IsAssignment, 93
PreprocessState, 93
Tang::AstNodeLibrary, 95
AstNodeLibrary, 97
compile, 98
compilePreprocess, 98
Default, 97
dump, 99
IsAssignment, 97
PreprocessState, 97
Tang::AstNodeMap, 99
AstNodeMap, 101
compile, 101
compilePreprocess, 102
Default, 101
dump, 103
IsAssignment, 101
PreprocessState, 101
Tang::AstNodePeriod, 103
AstNodePeriod, 105
compile, 106
compilePreprocess, 106
Default, 105
dump, 107
IsAssignment, 105
PreprocessState, 105
Tang::AstNodePrint, 107
AstNodePrint, 110
compile, 110
compilePreprocess, 111
Default, 110
dump, 111
IsAssignment, 110
PreprocessState, 109
Type, 110
Tang::AstNodeRangedFor, 112
AstNodeRangedFor, 114
compile, 114
compilePreprocess, 115
Default, 113
dump, 116
IsAssignment, 113
PreprocessState, 113
Tang::AstNodeReturn, 116
AstNodeReturn, 118
compile, 119
compilePreprocess, 119
Default, 118
dump, 120
IsAssignment, 118
PreprocessState, 118
Tang::AstNodeSlice, 120
AstNodeSlice, 123
compile, 123
compilePreprocess, 124
Default, 122
dump, 124
IsAssignment, 122
PreprocessState, 122
Tang::AstNodeString, 125
AstNodeString, 127, 128
compile, 128
compileLiteral, 129
compilePreprocess, 129
Default, 127
dump, 130
IsAssignment, 127
PreprocessState, 127
Tang::AstNodeTernary, 130
AstNodeTernary, 133
compile, 134
compilePreprocess, 134
Default, 133
dump, 135
IsAssignment, 133
PreprocessState, 133
Tang::AstNodeUnary, 135
AstNodeUnary, 138
compile, 138
compilePreprocess, 140
Default, 138
dump, 140
IsAssignment, 138
Negative, 138
Not, 138
Operator, 137
PreprocessState, 138
Tang::AstNodeUse, 141
AstNodeUse, 143
compile, 144

```

compilePreprocess, 144
Default, 143
dump, 145
IsAssignment, 143
PreprocessState, 143
Tang::AstNodeWhile, 145
  AstNodeWhile, 148
  compile, 149
  compilePreprocess, 149
  Default, 148
  dump, 150
  IsAssignment, 148
  PreprocessState, 148
Tang::ComputedExpression, 150
  __add, 153
  __asCode, 153
  __assign_index, 153
  __boolean, 154
  __divide, 154
  __equal, 154
  __float, 155
  __getIterator, 155
  __index, 155
  __integer, 156
  __iteratorNext, 156
  __lessThan, 156
  __modulo, 157
  __multiply, 157
  __negative, 158
  __not, 158
  __period, 158
  __slice, 159
  __string, 159
  __subtract, 159
  dump, 160
  is_equal, 160–162
  isCopyNeeded, 162
  makeCopy, 163
Tang::ComputedExpressionArray, 163
  __add, 166
  __asCode, 166
  __assign_index, 167
  __boolean, 167
  __divide, 168
  __equal, 168
  __float, 169
  __getIterator, 169
  __index, 169
  __integer, 170
  __iteratorNext, 170
  __lessThan, 170
  __modulo, 171
  __multiply, 171
  __negative, 172
  __not, 172
  __period, 172
  __slice, 173
  __string, 173
  __subtract, 174
  append, 174
  ComputedExpressionArray, 166
  dump, 175
  getContents, 175
  getMethods, 175
  is_equal, 175–177
  isCopyNeeded, 178
  makeCopy, 178
Tang::ComputedExpressionBoolean, 179
  __add, 181
  __asCode, 181
  __assign_index, 182
  __boolean, 182
  __divide, 182
  __equal, 183
  __float, 183
  __getIterator, 183
  __index, 184
  __integer, 184
  __iteratorNext, 184
  __lessThan, 185
  __modulo, 185
  __multiply, 185
  __negative, 186
  __not, 186
  __period, 186
  __slice, 187
  __string, 187
  __subtract, 187
  ComputedExpressionBoolean, 181
  dump, 188
  is_equal, 188–190
  isCopyNeeded, 190
  makeCopy, 191
Tang::ComputedExpressionCompiledFunction, 191
  __add, 194
  __asCode, 194
  __assign_index, 194
  __boolean, 195
  __divide, 195
  __equal, 196
  __float, 196
  __getIterator, 196
  __index, 197
  __integer, 197
  __iteratorNext, 197
  __lessThan, 198
  __modulo, 198
  __multiply, 198
  __negative, 199
  __not, 199
  __period, 199
  __slice, 200
  __string, 200
  __subtract, 200
  ComputedExpressionCompiledFunction, 193
  dump, 201

```

is_equal, 201–203
isCopyNeeded, 203
makeCopy, 204
Tang::ComputedExpressionError, 204
 __add, 207
 __asCode, 207
 __assign_index, 208
 __boolean, 208
 __divide, 208
 __equal, 209
 __float, 209
 __getIterator, 209
 __index, 210
 __integer, 210
 __iteratorNext, 210
 __lessThan, 211
 __modulo, 211
 __multiply, 211
 __negative, 212
 __not, 212
 __period, 212
 __slice, 213
 __string, 213
 __subtract, 213
ComputedExpressionError, 207
dump, 214
is_equal, 214–216
isCopyNeeded, 216
makeCopy, 217
Tang::ComputedExpressionFloat, 217
 __add, 220
 __asCode, 220
 __assign_index, 220
 __boolean, 221
 __divide, 221
 __equal, 222
 __float, 222
 __getIterator, 223
 __index, 223
 __integer, 223
 __iteratorNext, 223
 __lessThan, 224
 __modulo, 224
 __multiply, 225
 __negative, 225
 __not, 225
 __period, 226
 __slice, 226
 __string, 227
 __subtract, 227
ComputedExpressionFloat, 219
dump, 228
getValue, 228
is_equal, 228–230
isCopyNeeded, 230
makeCopy, 231
Tang::ComputedExpressionInteger, 231
 __add, 234
 __asCode, 234
 __assign_index, 234
 __boolean, 235
 __divide, 235
 __equal, 236
 __float, 236
 __getIterator, 237
 __index, 237
 __integer, 237
 __iteratorNext, 237
 __lessThan, 238
 __modulo, 238
 __multiply, 239
 __negative, 239
 __not, 240
 __period, 240
 __slice, 240
 __string, 241
 __subtract, 241
ComputedExpressionInteger, 233
dump, 242
getValue, 242
is_equal, 242–244
isCopyNeeded, 245
makeCopy, 245
Tang::ComputedExpressionIterator, 246
 __add, 248
 __asCode, 249
 __assign_index, 249
 __boolean, 249
 __divide, 250
 __equal, 250
 __float, 251
 __getIterator, 251
 __index, 251
 __integer, 252
 __iteratorNext, 252
 __lessThan, 252
 __modulo, 253
 __multiply, 253
 __negative, 254
 __not, 254
 __period, 254
 __slice, 255
 __string, 255
 __subtract, 255
ComputedExpressionIterator, 248
dump, 256
is_equal, 256, 258, 259
isCopyNeeded, 259
makeCopy, 259
Tang::ComputedExpressionIteratorEnd, 260
 __add, 262
 __asCode, 263
 __assign_index, 263
 __boolean, 263
 __divide, 264
 __equal, 264

```

__float, 265
__getIterator, 265
__index, 265
__integer, 266
__iteratorNext, 266
__lessThan, 266
__modulo, 267
__multiply, 267
__negative, 267
__not, 268
__period, 268
__slice, 268
__string, 269
__subtract, 269
dump, 270
is_equal, 270, 272, 273
isCopyNeeded, 273
makeCopy, 273
Tang::ComputedExpressionLibrary, 274
__add, 276
__asCode, 277
__assign_index, 277
__boolean, 277
__divide, 278
__equal, 278
__float, 279
__getIterator, 279
__index, 279
__integer, 280
__iteratorNext, 280
__lessThan, 280
__modulo, 281
__multiply, 281
__negative, 281
__not, 282
__period, 282
__slice, 282
__string, 283
__subtract, 283
dump, 284
is_equal, 284, 286, 287
isCopyNeeded, 287
makeCopy, 287
Tang::ComputedExpressionLibraryMath, 288
__add, 290
__asCode, 291
__assign_index, 291
__boolean, 292
__divide, 292
__equal, 292
__float, 293
__getIterator, 293
__index, 293
__integer, 294
__iteratorNext, 294
__lessThan, 294
__modulo, 295
__multiply, 295
__negative, 295
__not, 296
__period, 296
__slice, 296
__string, 297
__subtract, 297
dump, 298
getLibraryAttributes, 298
is_equal, 298–300
isCopyNeeded, 300
makeCopy, 301
Tang::ComputedExpressionMap, 301
__add, 304
__asCode, 304
__assign_index, 304
__boolean, 305
__divide, 305
__equal, 306
__float, 306
__getIterator, 306
__index, 307
__integer, 307
__iteratorNext, 307
__lessThan, 308
__modulo, 308
__multiply, 309
__negative, 309
__not, 309
__period, 309
__slice, 310
__string, 310
__subtract, 311
ComputedExpressionMap, 304
dump, 311
is_equal, 311–313
isCopyNeeded, 314
makeCopy, 314
Tang::ComputedExpressionNativeBoundFunction, 315
__add, 318
__asCode, 319
__assign_index, 319
__boolean, 319
__divide, 320
__equal, 320
__float, 321
__getIterator, 321
__index, 321
__integer, 322
__iteratorNext, 322
__lessThan, 322
__modulo, 324
__multiply, 324
__negative, 324
__not, 325
__period, 325
__slice, 325
__string, 326
__subtract, 326

```

ComputedExpressionNativeBoundFunction, 317
dump, 327
getArgc, 327
getFunction, 327
getTargetTypeIndex, 327
is_equal, 328–330
isCopyNeeded, 330
makeCopy, 330
Tang::ComputedExpressionNativeFunction, 331
__add, 333
__asCode, 334
__assign_index, 334
__boolean, 335
__divide, 335
__equal, 335
__float, 336
__getIterator, 336
__index, 337
__integer, 337
__iteratorNext, 337
__lessThan, 338
__modulo, 338
__multiply, 338
__negative, 339
__not, 339
__period, 339
__slice, 340
__string, 340
__subtract, 340
ComputedExpressionNativeFunction, 333
dump, 341
getArgc, 341
getFunction, 341
is_equal, 342–344
isCopyNeeded, 345
makeCopy, 345
Tang::ComputedExpressionNativeLibraryFunction, 346
__add, 348
__asCode, 349
__assign_index, 349
__boolean, 349
__divide, 350
__equal, 350
__float, 351
__getIterator, 351
__index, 351
__integer, 352
__iteratorNext, 352
__lessThan, 352
__modulo, 354
__multiply, 354
__negative, 354
__not, 355
__period, 355
__slice, 355
__string, 356
__subtract, 356
ComputedExpressionNativeLibraryFunction, 348
dump, 357
getFunction, 357
is_equal, 357–359
isCopyNeeded, 359
makeCopy, 360
Tang::ComputedExpressionString, 360
__add, 364
__asCode, 364
__assign_index, 365
__boolean, 365
__divide, 366
__equal, 366
__float, 367
__getIterator, 367
__index, 368
__integer, 368
__iteratorNext, 368
__lessThan, 369
__modulo, 370
__multiply, 370
__negative, 371
__not, 371
__period, 371
__slice, 372
__string, 373
__subtract, 373
bytesLength, 373
ComputedExpressionString, 363
dump, 373
getMethods, 374
getValue, 374
is_equal, 374–377
isCopyNeeded, 377
length, 377
makeCopy, 378
operator+=, 378
Tang::Context, 379
Tang::Error, 380
Error, 381
operator<<, 381
Tang::GarbageCollected, 382
~GarbageCollected, 385
GarbageCollected, 384, 385
isCopyNeeded, 385
make, 386
makeCopy, 386
operator!, 387
operator!=, 387
operator<, 392
operator<<, 397
operator<=, 392
operator>, 396
operator>=, 397
operator*, 388, 389
operator+, 389
operator-, 390
operator->, 391
operator/, 391

operator=, 393
operator==, 393–396
operator%, 388
Tang::HtmlEscape, 398
get_next_token, 400
HtmlEscape, 399
Tang::HtmlEscapeAscii, 400
get_next_token, 402
HtmlEscapeAscii, 401
Tang::location, 402
Tang::position, 404
Tang::Program, 405
addBreak, 408
addBytecode, 409
addContinue, 409
addIdentifier, 409
addIdentifierAssigned, 410
addLibraryAlias, 410
addString, 410
CodeType, 408
dumpBytecode, 411
execute, 411
functionsDeclared, 416
getAst, 412
getBytecode, 412
getCode, 412
getIdentifiers, 413
getIdentifiersAssigned, 413
getLibraryAliases, 413
getResult, 413
getStrings, 414
popBreakStack, 414
popContinueStack, 414
Program, 408
pushEnvironment, 415
Script, 408
setFunctionStackDeclaration, 415
setJumpTarget, 416
Template, 408
Tang::SingletonObjectPool< T >, 417
currentIndex, 419
currentRecycledIndex, 419
get, 418
getInstance, 418
recycle, 419
Tang::TangBase, 420
compileScript, 422
compileTemplate, 422
make_shared, 422
TangBase, 421
Tang::TangScanner, 423
get_next_token, 425
TangScanner, 424
Tang::Unescape, 425
get_next_token, 426
Unescape, 426
Tang::UnicodeString, 427
bytesLength, 429
length, 429
operator std::string, 429
operator<, 430
operator+, 430
operator+=, 430
operator==, 431
render, 431
renderAscii, 432
substr, 432
Trusted, 429
Type, 428
Untrusted, 429
TangBase
Tang::TangBase, 421
TangScanner
Tang::TangScanner, 424
Template
Tang::Program, 408
test/test.cpp, 533
test/testGarbageCollected.cpp, 534
test/testSingletonObjectPool.cpp, 535
test/testUnicodeString.cpp, 536
Trusted
Tang::UnicodeString, 429
Type
Tang::AstNodeCast, 49
Tang::AstNodePrint, 110
Tang::UnicodeString, 428
Unescape
Tang::Unescape, 426
unescape
unicodeString.hpp, 498
unicodeString.hpp
htmlEscape, 497
htmlEscapeAscii, 497
unescape, 498
Untrusted
Tang::UnicodeString, 429