

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	20
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::AstNodeMap Class Reference	95
5.19.1 Detailed Description	96
5.19.2 Member Enumeration Documentation	96
5.19.2.1 PreprocessState	96
5.19.3 Constructor & Destructor Documentation	97
5.19.3.1 AstNodeMap()	97
5.19.4 Member Function Documentation	97
5.19.4.1 compile()	97
5.19.4.2 compilePreprocess()	98
5.19.4.3 dump()	98
5.20 Tang::AstNodePeriod 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 AstNodePeriod()	101
5.20.4 Member Function Documentation	102
5.20.4.1 compile()	102
5.20.4.2 compilePreprocess()	102
5.20.4.3 dump()	103
5.21 Tang::AstNodePrint Class Reference	103
5.21.1 Detailed Description	105
5.21.2 Member Enumeration Documentation	105
5.21.2.1 PreprocessState	105
5.21.2.2 Type	106
5.21.3 Constructor & Destructor Documentation	106
5.21.3.1 AstNodePrint()	106
5.21.4 Member Function Documentation	106
5.21.4.1 compile()	106
5.21.4.2 compilePreprocess()	107
5.21.4.3 dump()	107
5.22 Tang::AstNodeRangedFor Class Reference	108
5.22.1 Detailed Description	109
5.22.2 Member Enumeration Documentation	109
5.22.2.1 PreprocessState	109
5.22.3 Constructor & Destructor Documentation	110
5.22.3.1 AstNodeRangedFor()	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()	112
5.23 Tang::AstNodeReturn Class Reference	112
5.23.1 Detailed Description	114
5.23.2 Member Enumeration Documentation	114
5.23.2.1 PreprocessState	114
5.23.3 Constructor & Destructor Documentation	114
5.23.3.1 AstNodeReturn()	114
5.23.4 Member Function Documentation	115
5.23.4.1 compile()	115
5.23.4.2 compilePreprocess()	115
5.23.4.3 dump()	116
5.24 Tang::AstNodeSlice 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	119
5.24.3.1 AstNodeSlice()	119
5.24.4 Member Function Documentation	119
5.24.4.1 compile()	119
5.24.4.2 compilePreprocess()	120
5.24.4.3 dump()	120
5.25 Tang::AstNodeString Class Reference	121
5.25.1 Detailed Description	123
5.25.2 Member Enumeration Documentation	123
5.25.2.1 PreprocessState	123
5.25.3 Constructor & Destructor Documentation	123
5.25.3.1 AstNodeString()	123
5.25.4 Member Function Documentation	124
5.25.4.1 compile()	124
5.25.4.2 compileLiteral()	124
5.25.4.3 compilePreprocess()	125
5.25.4.4 dump()	125
5.26 Tang::AstNodeTernary Class Reference	126
5.26.1 Detailed Description	128
5.26.2 Member Enumeration Documentation	128
5.26.2.1 PreprocessState	128
5.26.3 Constructor & Destructor Documentation	128
5.26.3.1 AstNodeTernary()	129
5.26.4 Member Function Documentation	129
5.26.4.1 compile()	129
5.26.4.2 compilePreprocess()	130
5.26.4.3 dump()	130
5.27 Tang::AstNodeUnary Class Reference	130
5.27.1 Detailed Description	132
5.27.2 Member Enumeration Documentation	132
5.27.2.1 Operator	132
5.27.2.2 PreprocessState	133
5.27.3 Constructor & Destructor Documentation	133
5.27.3.1 AstNodeUnary()	133
5.27.4 Member Function Documentation	133
5.27.4.1 compile()	133
5.27.4.2 compilePreprocess()	135
5.27.4.3 dump()	135
5.28 Tang::AstNodeWhile Class Reference	136
5.28.1 Detailed Description	138
5.28.2 Member Enumeration Documentation	138

5.28.2.1 PreprocessState	138
5.28.3 Constructor & Destructor Documentation	138
5.28.3.1 AstNodeWhile()	138
5.28.4 Member Function Documentation	139
5.28.4.1 compile()	139
5.28.4.2 compilePreprocess()	140
5.28.4.3 dump()	140
5.29 Tang::ComputedExpression Class Reference	140
5.29.1 Detailed Description	143
5.29.2 Member Function Documentation	143
5.29.2.1 __add()	143
5.29.2.2 __asCode()	143
5.29.2.3 __assign_index()	144
5.29.2.4 __boolean()	144
5.29.2.5 __divide()	144
5.29.2.6 __equal()	145
5.29.2.7 __float()	145
5.29.2.8 __getIterator()	145
5.29.2.9 __index()	146
5.29.2.10 __integer()	146
5.29.2.11 __iteratorNext()	146
5.29.2.12 __lessThan()	147
5.29.2.13 __modulo()	147
5.29.2.14 __multiply()	148
5.29.2.15 __negative()	148
5.29.2.16 __not()	148
5.29.2.17 __period()	148
5.29.2.18 __slice()	149
5.29.2.19 __string()	149
5.29.2.20 __subtract()	150
5.29.2.21 dump()	150
5.29.2.22 is_equal() [1/6]	150
5.29.2.23 is_equal() [2/6]	151
5.29.2.24 is_equal() [3/6]	151
5.29.2.25 is_equal() [4/6]	151
5.29.2.26 is_equal() [5/6]	152
5.29.2.27 is_equal() [6/6]	152
5.29.2.28 isCopyNeeded()	153
5.29.2.29 makeCopy()	153
5.30 Tang::ComputedExpressionArray Class Reference	153
5.30.1 Detailed Description	156
5.30.2 Constructor & Destructor Documentation	156

5.30.2.1 <code>ComputedExpressionArray()</code>	156
5.30.3 Member Function Documentation	157
5.30.3.1 <code>__add()</code>	157
5.30.3.2 <code>__asCode()</code>	157
5.30.3.3 <code>__assign_index()</code>	157
5.30.3.4 <code>__boolean()</code>	158
5.30.3.5 <code>__divide()</code>	158
5.30.3.6 <code>__equal()</code>	159
5.30.3.7 <code>__float()</code>	159
5.30.3.8 <code>__getIterator()</code>	159
5.30.3.9 <code>__index()</code>	160
5.30.3.10 <code>__integer()</code>	160
5.30.3.11 <code>__iteratorNext()</code>	161
5.30.3.12 <code>__lessThan()</code>	161
5.30.3.13 <code>__modulo()</code>	161
5.30.3.14 <code>__multiply()</code>	162
5.30.3.15 <code>__negative()</code>	162
5.30.3.16 <code>__not()</code>	162
5.30.3.17 <code>__period()</code>	162
5.30.3.18 <code>__slice()</code>	163
5.30.3.19 <code>__string()</code>	164
5.30.3.20 <code>__subtract()</code>	164
5.30.3.21 <code>append()</code>	164
5.30.3.22 <code>dump()</code>	165
5.30.3.23 <code>getContents()</code>	165
5.30.3.24 <code>getMethods()</code>	165
5.30.3.25 <code>is_equal()</code> [1/6]	166
5.30.3.26 <code>is_equal()</code> [2/6]	166
5.30.3.27 <code>is_equal()</code> [3/6]	167
5.30.3.28 <code>is_equal()</code> [4/6]	167
5.30.3.29 <code>is_equal()</code> [5/6]	168
5.30.3.30 <code>is_equal()</code> [6/6]	168
5.30.3.31 <code>isCopyNeeded()</code>	168
5.30.3.32 <code>makeCopy()</code>	169
5.31 <code>Tang::ComputedExpressionBoolean</code> Class Reference	169
5.31.1 Detailed Description	171
5.31.2 Constructor & Destructor Documentation	171
5.31.2.1 <code>ComputedExpressionBoolean()</code>	171
5.31.3 Member Function Documentation	172
5.31.3.1 <code>__add()</code>	172
5.31.3.2 <code>__asCode()</code>	172
5.31.3.3 <code>__assign_index()</code>	172

5.31.3.4 <code>__boolean()</code>	173
5.31.3.5 <code>__divide()</code>	173
5.31.3.6 <code>__equal()</code>	173
5.31.3.7 <code>__float()</code>	174
5.31.3.8 <code>__getIterator()</code>	174
5.31.3.9 <code>__index()</code>	175
5.31.3.10 <code>__integer()</code>	175
5.31.3.11 <code>__iteratorNext()</code>	175
5.31.3.12 <code>__lessThan()</code>	176
5.31.3.13 <code>__modulo()</code>	176
5.31.3.14 <code>__multiply()</code>	176
5.31.3.15 <code>__negative()</code>	177
5.31.3.16 <code>__not()</code>	177
5.31.3.17 <code>__period()</code>	177
5.31.3.18 <code>__slice()</code>	178
5.31.3.19 <code>__string()</code>	178
5.31.3.20 <code>__subtract()</code>	179
5.31.3.21 <code>dump()</code>	179
5.31.3.22 <code>is_equal()</code> [1/6]	179
5.31.3.23 <code>is_equal()</code> [2/6]	180
5.31.3.24 <code>is_equal()</code> [3/6]	180
5.31.3.25 <code>is_equal()</code> [4/6]	180
5.31.3.26 <code>is_equal()</code> [5/6]	181
5.31.3.27 <code>is_equal()</code> [6/6]	181
5.31.3.28 <code>isCopyNeeded()</code>	181
5.31.3.29 <code>makeCopy()</code>	182
5.32 Tang::ComputedExpressionCompiledFunction Class Reference	182
5.32.1 Detailed Description	185
5.32.2 Constructor & Destructor Documentation	185
5.32.2.1 <code>ComputedExpressionCompiledFunction()</code>	185
5.32.3 Member Function Documentation	185
5.32.3.1 <code>__add()</code>	185
5.32.3.2 <code>__asCode()</code>	186
5.32.3.3 <code>__assign_index()</code>	186
5.32.3.4 <code>__boolean()</code>	186
5.32.3.5 <code>__divide()</code>	186
5.32.3.6 <code>__equal()</code>	187
5.32.3.7 <code>__float()</code>	187
5.32.3.8 <code>__getIterator()</code>	188
5.32.3.9 <code>__index()</code>	188
5.32.3.10 <code>__integer()</code>	188
5.32.3.11 <code>__iteratorNext()</code>	188

5.32.3.12 <code>__lessThan()</code>	189
5.32.3.13 <code>__modulo()</code>	189
5.32.3.14 <code>__multiply()</code>	190
5.32.3.15 <code>__negative()</code>	190
5.32.3.16 <code>__not()</code>	190
5.32.3.17 <code>__period()</code>	190
5.32.3.18 <code>__slice()</code>	191
5.32.3.19 <code>__string()</code>	191
5.32.3.20 <code>__subtract()</code>	192
5.32.3.21 <code>dump()</code>	192
5.32.3.22 <code>is_equal()</code> [1/6]	192
5.32.3.23 <code>is_equal()</code> [2/6]	193
5.32.3.24 <code>is_equal()</code> [3/6]	193
5.32.3.25 <code>is_equal()</code> [4/6]	193
5.32.3.26 <code>is_equal()</code> [5/6]	194
5.32.3.27 <code>is_equal()</code> [6/6]	194
5.32.3.28 <code>isCopyNeeded()</code>	194
5.32.3.29 <code>makeCopy()</code>	195
5.33 <code>Tangi::ComputedExpressionError</code> Class Reference	195
5.33.1 Detailed Description	198
5.33.2 Constructor & Destructor Documentation	198
5.33.2.1 <code>ComputedExpressionError()</code>	198
5.33.3 Member Function Documentation	198
5.33.3.1 <code>__add()</code>	198
5.33.3.2 <code>__asCode()</code>	199
5.33.3.3 <code>__assign_index()</code>	199
5.33.3.4 <code>__boolean()</code>	199
5.33.3.5 <code>__divide()</code>	200
5.33.3.6 <code>__equal()</code>	200
5.33.3.7 <code>__float()</code>	200
5.33.3.8 <code>__getIterator()</code>	201
5.33.3.9 <code>__index()</code>	201
5.33.3.10 <code>__integer()</code>	201
5.33.3.11 <code>__iteratorNext()</code>	202
5.33.3.12 <code>__lessThan()</code>	202
5.33.3.13 <code>__modulo()</code>	202
5.33.3.14 <code>__multiply()</code>	203
5.33.3.15 <code>__negative()</code>	203
5.33.3.16 <code>__not()</code>	203
5.33.3.17 <code>__period()</code>	203
5.33.3.18 <code>__slice()</code>	204
5.33.3.19 <code>__string()</code>	204

5.33.3.20 <code>__subtract()</code>	205
5.33.3.21 <code>dump()</code>	206
5.33.3.22 <code>is_equal()</code> [1/6]	206
5.33.3.23 <code>is_equal()</code> [2/6]	207
5.33.3.24 <code>is_equal()</code> [3/6]	208
5.33.3.25 <code>is_equal()</code> [4/6]	208
5.33.3.26 <code>is_equal()</code> [5/6]	209
5.33.3.27 <code>is_equal()</code> [6/6]	209
5.33.3.28 <code>isCopyNeeded()</code>	209
5.33.3.29 <code>makeCopy()</code>	210
5.34 Tang::ComputedExpressionFloat Class Reference	210
5.34.1 Detailed Description	212
5.34.2 Constructor & Destructor Documentation	212
5.34.2.1 ComputedExpressionFloat()	212
5.34.3 Member Function Documentation	213
5.34.3.1 <code>__add()</code>	213
5.34.3.2 <code>__asCode()</code>	213
5.34.3.3 <code>__assign_index()</code>	213
5.34.3.4 <code>__boolean()</code>	214
5.34.3.5 <code>__divide()</code>	214
5.34.3.6 <code>__equal()</code>	215
5.34.3.7 <code>__float()</code>	215
5.34.3.8 <code>__getIterator()</code>	216
5.34.3.9 <code>__index()</code>	216
5.34.3.10 <code>__integer()</code>	216
5.34.3.11 <code>__iteratorNext()</code>	216
5.34.3.12 <code>__lessThan()</code>	217
5.34.3.13 <code>__modulo()</code>	217
5.34.3.14 <code>__multiply()</code>	218
5.34.3.15 <code>__negative()</code>	218
5.34.3.16 <code>__not()</code>	219
5.34.3.17 <code>__period()</code>	219
5.34.3.18 <code>__slice()</code>	219
5.34.3.19 <code>__string()</code>	220
5.34.3.20 <code>__subtract()</code>	220
5.34.3.21 <code>dump()</code>	221
5.34.3.22 <code>getValue()</code>	221
5.34.3.23 <code>is_equal()</code> [1/6]	221
5.34.3.24 <code>is_equal()</code> [2/6]	222
5.34.3.25 <code>is_equal()</code> [3/6]	222
5.34.3.26 <code>is_equal()</code> [4/6]	222
5.34.3.27 <code>is_equal()</code> [5/6]	223

5.34.3.28 <code>is_equal()</code> [6/6]	223
5.34.3.29 <code>isCopyNeeded()</code>	224
5.34.3.30 <code>makeCopy()</code>	224
5.35 <code>Tang::ComputedExpressionInteger</code> Class Reference	224
5.35.1 Detailed Description	226
5.35.2 Constructor & Destructor Documentation	227
5.35.2.1 <code>ComputedExpressionInteger()</code>	227
5.35.3 Member Function Documentation	227
5.35.3.1 <code>__add()</code>	227
5.35.3.2 <code>__asCode()</code>	228
5.35.3.3 <code>__assign_index()</code>	228
5.35.3.4 <code>__boolean()</code>	228
5.35.3.5 <code>__divide()</code>	228
5.35.3.6 <code>__equal()</code>	229
5.35.3.7 <code>__float()</code>	230
5.35.3.8 <code>__getIterator()</code>	230
5.35.3.9 <code>__index()</code>	230
5.35.3.10 <code>__integer()</code>	231
5.35.3.11 <code>__iteratorNext()</code>	231
5.35.3.12 <code>__lessThan()</code>	231
5.35.3.13 <code>__modulo()</code>	232
5.35.3.14 <code>__multiply()</code>	232
5.35.3.15 <code>__negative()</code>	233
5.35.3.16 <code>__not()</code>	233
5.35.3.17 <code>__period()</code>	233
5.35.3.18 <code>__slice()</code>	234
5.35.3.19 <code>__string()</code>	234
5.35.3.20 <code>__subtract()</code>	235
5.35.3.21 <code>dump()</code>	236
5.35.3.22 <code>getValue()</code>	236
5.35.3.23 <code>is_equal()</code> [1/6]	236
5.35.3.24 <code>is_equal()</code> [2/6]	236
5.35.3.25 <code>is_equal()</code> [3/6]	237
5.35.3.26 <code>is_equal()</code> [4/6]	237
5.35.3.27 <code>is_equal()</code> [5/6]	238
5.35.3.28 <code>is_equal()</code> [6/6]	238
5.35.3.29 <code>isCopyNeeded()</code>	238
5.35.3.30 <code>makeCopy()</code>	239
5.36 <code>Tang::ComputedExpressionIterator</code> Class Reference	239
5.36.1 Detailed Description	242
5.36.2 Constructor & Destructor Documentation	242
5.36.2.1 <code>ComputedExpressionIterator()</code>	242

5.36.3 Member Function Documentation	242
5.36.3.1 <code>__add()</code>	242
5.36.3.2 <code>__asCode()</code>	243
5.36.3.3 <code>__assign_index()</code>	243
5.36.3.4 <code>__boolean()</code>	243
5.36.3.5 <code>__divide()</code>	243
5.36.3.6 <code>__equal()</code>	244
5.36.3.7 <code>__float()</code>	244
5.36.3.8 <code>__getIterator()</code>	244
5.36.3.9 <code>__index()</code>	245
5.36.3.10 <code>__integer()</code>	245
5.36.3.11 <code>__iteratorNext()</code>	245
5.36.3.12 <code>__lessThan()</code>	246
5.36.3.13 <code>__modulo()</code>	246
5.36.3.14 <code>__multiply()</code>	247
5.36.3.15 <code>__negative()</code>	247
5.36.3.16 <code>__not()</code>	247
5.36.3.17 <code>__period()</code>	248
5.36.3.18 <code>__slice()</code>	248
5.36.3.19 <code>__string()</code>	248
5.36.3.20 <code>__subtract()</code>	249
5.36.3.21 <code>dump()</code>	249
5.36.3.22 <code>is_equal()</code> [1/6]	249
5.36.3.23 <code>is_equal()</code> [2/6]	250
5.36.3.24 <code>is_equal()</code> [3/6]	250
5.36.3.25 <code>is_equal()</code> [4/6]	251
5.36.3.26 <code>is_equal()</code> [5/6]	251
5.36.3.27 <code>is_equal()</code> [6/6]	251
5.36.3.28 <code>isCopyNeeded()</code>	252
5.36.3.29 <code>makeCopy()</code>	252
5.37 <code>Tang::ComputedExpressionIteratorEnd</code> Class Reference	253
5.37.1 Detailed Description	255
5.37.2 Member Function Documentation	255
5.37.2.1 <code>__add()</code>	255
5.37.2.2 <code>__asCode()</code>	255
5.37.2.3 <code>__assign_index()</code>	256
5.37.2.4 <code>__boolean()</code>	256
5.37.2.5 <code>__divide()</code>	256
5.37.2.6 <code>__equal()</code>	257
5.37.2.7 <code>__float()</code>	257
5.37.2.8 <code>__getIterator()</code>	257
5.37.2.9 <code>__index()</code>	258

5.37.2.10 __integer()	258
5.37.2.11 __iteratorNext()	258
5.37.2.12 __lessThan()	259
5.37.2.13 __modulo()	259
5.37.2.14 __multiply()	260
5.37.2.15 __negative()	260
5.37.2.16 __not()	260
5.37.2.17 __period()	260
5.37.2.18 __slice()	261
5.37.2.19 __string()	261
5.37.2.20 __subtract()	262
5.37.2.21 dump()	263
5.37.2.22 is_equal() [1/6]	263
5.37.2.23 is_equal() [2/6]	264
5.37.2.24 is_equal() [3/6]	265
5.37.2.25 is_equal() [4/6]	265
5.37.2.26 is_equal() [5/6]	266
5.37.2.27 is_equal() [6/6]	266
5.37.2.28 isCopyNeeded()	266
5.37.2.29 makeCopy()	267
5.38 Tang::ComputedExpressionMap Class Reference	267
5.38.1 Detailed Description	270
5.38.2 Constructor & Destructor Documentation	270
5.38.2.1 ComputedExpressionMap()	270
5.38.3 Member Function Documentation	270
5.38.3.1 __add()	270
5.38.3.2 __asCode()	271
5.38.3.3 __assign_index()	271
5.38.3.4 __boolean()	272
5.38.3.5 __divide()	272
5.38.3.6 __equal()	272
5.38.3.7 __float()	273
5.38.3.8 __getIterator()	273
5.38.3.9 __index()	273
5.38.3.10 __integer()	274
5.38.3.11 __iteratorNext()	274
5.38.3.12 __lessThan()	275
5.38.3.13 __modulo()	275
5.38.3.14 __multiply()	275
5.38.3.15 __negative()	276
5.38.3.16 __not()	276
5.38.3.17 __period()	276

5.38.3.18 <code>__slice()</code>	277
5.38.3.19 <code>__string()</code>	277
5.38.3.20 <code>__subtract()</code>	278
5.38.3.21 <code>dump()</code>	278
5.38.3.22 <code>is_equal()</code> [1/6]	278
5.38.3.23 <code>is_equal()</code> [2/6]	279
5.38.3.24 <code>is_equal()</code> [3/6]	279
5.38.3.25 <code>is_equal()</code> [4/6]	279
5.38.3.26 <code>is_equal()</code> [5/6]	280
5.38.3.27 <code>is_equal()</code> [6/6]	280
5.38.3.28 <code>isCopyNeeded()</code>	280
5.38.3.29 <code>makeCopy()</code>	281
5.39 <code>Tang::ComputedExpressionNativeBoundFunction</code> Class Reference	281
5.39.1 Detailed Description	284
5.39.2 Constructor & Destructor Documentation	284
5.39.2.1 <code>ComputedExpressionNativeBoundFunction()</code>	284
5.39.3 Member Function Documentation	284
5.39.3.1 <code>__add()</code>	284
5.39.3.2 <code>__asCode()</code>	285
5.39.3.3 <code>__assign_index()</code>	285
5.39.3.4 <code>__boolean()</code>	285
5.39.3.5 <code>__divide()</code>	285
5.39.3.6 <code>__equal()</code>	286
5.39.3.7 <code>__float()</code>	286
5.39.3.8 <code>__getIterator()</code>	287
5.39.3.9 <code>__index()</code>	287
5.39.3.10 <code>__integer()</code>	287
5.39.3.11 <code>__iteratorNext()</code>	288
5.39.3.12 <code>__lessThan()</code>	288
5.39.3.13 <code>__modulo()</code>	288
5.39.3.14 <code>__multiply()</code>	289
5.39.3.15 <code>__negative()</code>	289
5.39.3.16 <code>__not()</code>	290
5.39.3.17 <code>__period()</code>	290
5.39.3.18 <code>__slice()</code>	290
5.39.3.19 <code>__string()</code>	291
5.39.3.20 <code>__subtract()</code>	291
5.39.3.21 <code>dump()</code>	291
5.39.3.22 <code>is_equal()</code> [1/6]	292
5.39.3.23 <code>is_equal()</code> [2/6]	292
5.39.3.24 <code>is_equal()</code> [3/6]	292
5.39.3.25 <code>is_equal()</code> [4/6]	293

5.39.3.26 <code>is_equal()</code> [5/6]	293
5.39.3.27 <code>is_equal()</code> [6/6]	293
5.39.3.28 <code>isCopyNeeded()</code>	294
5.39.3.29 <code>makeCopy()</code>	294
5.40 <code>Tang::ComputedExpressionString</code> Class Reference	295
5.40.1 Detailed Description	297
5.40.2 Constructor & Destructor Documentation	297
5.40.2.1 <code>ComputedExpressionString()</code>	297
5.40.3 Member Function Documentation	298
5.40.3.1 <code>__add()</code>	298
5.40.3.2 <code>__asCode()</code>	298
5.40.3.3 <code>__assign_index()</code>	299
5.40.3.4 <code>__boolean()</code>	299
5.40.3.5 <code>__divide()</code>	300
5.40.3.6 <code>__equal()</code>	300
5.40.3.7 <code>__float()</code>	301
5.40.3.8 <code>__getIterator()</code>	301
5.40.3.9 <code>__index()</code>	302
5.40.3.10 <code>__integer()</code>	302
5.40.3.11 <code>__iteratorNext()</code>	302
5.40.3.12 <code>__lessThan()</code>	303
5.40.3.13 <code>__modulo()</code>	304
5.40.3.14 <code>__multiply()</code>	304
5.40.3.15 <code>__negative()</code>	304
5.40.3.16 <code>__not()</code>	305
5.40.3.17 <code>__period()</code>	305
5.40.3.18 <code>__slice()</code>	305
5.40.3.19 <code>__string()</code>	306
5.40.3.20 <code>__subtract()</code>	306
5.40.3.21 <code>dump()</code>	307
5.40.3.22 <code>getMethods()</code>	307
5.40.3.23 <code>getValue()</code>	308
5.40.3.24 <code>is_equal()</code> [1/6]	308
5.40.3.25 <code>is_equal()</code> [2/6]	308
5.40.3.26 <code>is_equal()</code> [3/6]	309
5.40.3.27 <code>is_equal()</code> [4/6]	309
5.40.3.28 <code>is_equal()</code> [5/6]	310
5.40.3.29 <code>is_equal()</code> [6/6]	310
5.40.3.30 <code>isCopyNeeded()</code>	310
5.40.3.31 <code>makeCopy()</code>	311
5.41 <code>Tang::Error</code> Class Reference	311
5.41.1 Detailed Description	312

5.41.2 Constructor & Destructor Documentation	312
5.41.2.1 Error() [1/2]	312
5.41.2.2 Error() [2/2]	312
5.41.3 Friends And Related Function Documentation	313
5.41.3.1 operator<<	313
5.42 Tang::GarbageCollected Class Reference	313
5.42.1 Detailed Description	316
5.42.2 Constructor & Destructor Documentation	316
5.42.2.1 GarbageCollected() [1/3]	316
5.42.2.2 GarbageCollected() [2/3]	316
5.42.2.3 ~GarbageCollected()	316
5.42.2.4 GarbageCollected() [3/3]	317
5.42.3 Member Function Documentation	317
5.42.3.1 isCopyNeeded()	317
5.42.3.2 make()	317
5.42.3.3 makeCopy()	318
5.42.3.4 operator"!"()	318
5.42.3.5 operator"!!="()	319
5.42.3.6 operator%()	319
5.42.3.7 operator*() [1/2]	320
5.42.3.8 operator*() [2/2]	320
5.42.3.9 operator+()	321
5.42.3.10 operator-() [1/2]	321
5.42.3.11 operator-() [2/2]	322
5.42.3.12 operator->()	322
5.42.3.13 operator/()	323
5.42.3.14 operator<()	323
5.42.3.15 operator<=()	324
5.42.3.16 operator=() [1/2]	324
5.42.3.17 operator=() [2/2]	324
5.42.3.18 operator==() [1/8]	326
5.42.3.19 operator==() [2/8]	326
5.42.3.20 operator==() [3/8]	326
5.42.3.21 operator==() [4/8]	327
5.42.3.22 operator==() [5/8]	327
5.42.3.23 operator==() [6/8]	328
5.42.3.24 operator==() [7/8]	328
5.42.3.25 operator==() [8/8]	328
5.42.3.26 operator>()	329
5.42.3.27 operator>=()	329
5.42.4 Friends And Related Function Documentation	330
5.42.4.1 operator<<	330

5.43 Tang::HtmlEscape Class Reference	330
5.43.1 Detailed Description	331
5.43.2 Constructor & Destructor Documentation	331
5.43.2.1 HtmlEscape()	332
5.43.3 Member Function Documentation	332
5.43.3.1 get_next_token()	332
5.44 Tang::HtmlEscapeAscii Class Reference	332
5.44.1 Detailed Description	333
5.44.2 Constructor & Destructor Documentation	333
5.44.2.1 HtmlEscapeAscii()	334
5.44.3 Member Function Documentation	334
5.44.3.1 get_next_token()	334
5.45 Tang::location Class Reference	334
5.45.1 Detailed Description	336
5.46 Tang::position Class Reference	336
5.46.1 Detailed Description	337
5.47 Tang::Program Class Reference	337
5.47.1 Detailed Description	340
5.47.2 Member Enumeration Documentation	340
5.47.2.1 CodeType	340
5.47.3 Constructor & Destructor Documentation	340
5.47.3.1 Program()	340
5.47.4 Member Function Documentation	340
5.47.4.1 addBreak()	341
5.47.4.2 addBytecode()	341
5.47.4.3 addContinue()	341
5.47.4.4 addIdentifier()	341
5.47.4.5 addIdentifierAssigned()	342
5.47.4.6 addString()	342
5.47.4.7 dumpBytecode()	342
5.47.4.8 execute()	343
5.47.4.9 getAst()	343
5.47.4.10 getBytecode()	343
5.47.4.11 getCode()	344
5.47.4.12 getIdentifiers()	344
5.47.4.13 getIdentifiersAssigned()	344
5.47.4.14 getResult()	344
5.47.4.15 getStrings()	345
5.47.4.16 popBreakStack()	345
5.47.4.17 popContinueStack()	345
5.47.4.18 pushEnvironment()	346
5.47.4.19 setFunctionStackDeclaration()	346

5.47.4.20 setJumpTarget()	347
5.47.5 Member Data Documentation	347
5.47.5.1 functionsDeclared	347
5.48 Tang::SingletonObjectPool< T > Class Template Reference	348
5.48.1 Detailed Description	349
5.48.2 Member Function Documentation	349
5.48.2.1 get()	349
5.48.2.2 getInstance()	350
5.48.2.3 recycle()	350
5.48.3 Member Data Documentation	350
5.48.3.1 currentIndex	350
5.48.3.2 currentRecycledIndex	350
5.49 Tang::TangBase Class Reference	351
5.49.1 Detailed Description	352
5.49.2 Constructor & Destructor Documentation	352
5.49.2.1 TangBase()	352
5.49.3 Member Function Documentation	352
5.49.3.1 compileScript()	352
5.49.3.2 make_shared()	353
5.50 Tang::TangScanner Class Reference	353
5.50.1 Detailed Description	354
5.50.2 Constructor & Destructor Documentation	355
5.50.2.1 TangScanner()	355
5.50.3 Member Function Documentation	355
5.50.3.1 get_next_token()	355
5.51 Tang::Unescape Class Reference	356
5.51.1 Detailed Description	356
5.51.2 Constructor & Destructor Documentation	357
5.51.2.1 Unescape()	357
5.51.3 Member Function Documentation	357
5.51.3.1 get_next_token()	357
5.52 Tang::UnicodeString Class Reference	358
5.52.1 Detailed Description	359
5.52.2 Constructor & Destructor Documentation	359
5.52.2.1 UnicodeString()	359
5.52.3 Member Function Documentation	359
5.52.3.1 bytesLength()	359
5.52.3.2 length()	360
5.52.3.3 operator std::string()	360
5.52.3.4 operator+()	360
5.52.3.5 operator<()	361
5.52.3.6 operator==()	361

5.52.3.7 substr()	361
6 File Documentation	363
6.1 build/generated/location.hh File Reference	363
6.1.1 Detailed Description	364
6.1.2 Function Documentation	364
6.1.2.1 operator<<() [1/2]	364
6.1.2.2 operator<<() [2/2]	365
6.2 include/astNode.hpp File Reference	365
6.2.1 Detailed Description	366
6.3 include/astNodeArray.hpp File Reference	366
6.3.1 Detailed Description	367
6.4 include/astNodeAssign.hpp File Reference	367
6.4.1 Detailed Description	368
6.5 include/astNodeBinary.hpp File Reference	368
6.5.1 Detailed Description	369
6.6 include/astNodeBlock.hpp File Reference	369
6.6.1 Detailed Description	370
6.7 include/astNodeBoolean.hpp File Reference	370
6.7.1 Detailed Description	371
6.8 include/astNodeBreak.hpp File Reference	371
6.8.1 Detailed Description	372
6.9 include/astNodeCast.hpp File Reference	372
6.9.1 Detailed Description	373
6.10 include/astNodeContinue.hpp File Reference	373
6.10.1 Detailed Description	374
6.11 include/astNodeDoWhile.hpp File Reference	374
6.11.1 Detailed Description	375
6.12 include/astNodeFloat.hpp File Reference	375
6.12.1 Detailed Description	376
6.13 include/astNodeFor.hpp File Reference	376
6.13.1 Detailed Description	377
6.14 include/astNodeFunctionCall.hpp File Reference	377
6.14.1 Detailed Description	378
6.15 include/astNodeFunctionDeclaration.hpp File Reference	378
6.15.1 Detailed Description	379
6.16 include/astNodeIdentifier.hpp File Reference	379
6.16.1 Detailed Description	380
6.17 include/astNodeIfElse.hpp File Reference	380
6.17.1 Detailed Description	381
6.18 include/astNodeIndex.hpp File Reference	381
6.18.1 Detailed Description	382

6.19 include/astNodeInteger.hpp File Reference	382
6.19.1 Detailed Description	383
6.20 include/astNodeMap.hpp File Reference	383
6.20.1 Detailed Description	384
6.21 include/astNodePeriod.hpp File Reference	384
6.21.1 Detailed Description	385
6.22 include/astNodePrint.hpp File Reference	385
6.22.1 Detailed Description	386
6.23 include/astNodeRangedFor.hpp File Reference	386
6.23.1 Detailed Description	387
6.24 include/astNodeReturn.hpp File Reference	387
6.24.1 Detailed Description	388
6.25 include/astNodeSlice.hpp File Reference	388
6.25.1 Detailed Description	389
6.26 include/astNodeString.hpp File Reference	389
6.26.1 Detailed Description	389
6.27 include/astNodeTernary.hpp File Reference	390
6.27.1 Detailed Description	390
6.28 include/astNodeUnary.hpp File Reference	391
6.28.1 Detailed Description	391
6.29 include/astNodeWhile.hpp File Reference	392
6.29.1 Detailed Description	392
6.30 include/computedExpression.hpp File Reference	393
6.30.1 Detailed Description	393
6.31 include/computedExpressionArray.hpp File Reference	394
6.31.1 Detailed Description	394
6.32 include/computedExpressionBoolean.hpp File Reference	395
6.32.1 Detailed Description	395
6.33 include/computedExpressionCompiledFunction.hpp File Reference	395
6.33.1 Detailed Description	396
6.34 include/computedExpressionError.hpp File Reference	396
6.34.1 Detailed Description	397
6.35 include/computedExpressionFloat.hpp File Reference	397
6.35.1 Detailed Description	398
6.36 include/computedExpressionInteger.hpp File Reference	398
6.36.1 Detailed Description	399
6.37 include/computedExpressionIterator.hpp File Reference	399
6.37.1 Detailed Description	400
6.38 include/computedExpressionIteratorEnd.hpp File Reference	400
6.38.1 Detailed Description	401
6.39 include/computedExpressionMap.hpp File Reference	401
6.39.1 Detailed Description	402

6.40 include/computedExpressionNativeBoundFunction.hpp File Reference	402
6.40.1 Detailed Description	403
6.41 include/computedExpressionString.hpp File Reference	403
6.41.1 Detailed Description	404
6.42 include/error.hpp File Reference	404
6.42.1 Detailed Description	405
6.43 include/garbageCollected.hpp File Reference	405
6.43.1 Detailed Description	406
6.44 include/htmlEscape.hpp File Reference	406
6.44.1 Detailed Description	407
6.45 include/htmlEscapeAscii.hpp File Reference	407
6.45.1 Detailed Description	408
6.46 include/macros.hpp File Reference	408
6.46.1 Detailed Description	409
6.47 include/opcode.hpp File Reference	409
6.47.1 Detailed Description	410
6.47.2 Enumeration Type Documentation	410
6.47.2.1 Opcode	410
6.48 include/program.hpp File Reference	411
6.48.1 Detailed Description	412
6.49 include/singleObjectPool.hpp File Reference	412
6.49.1 Detailed Description	413
6.50 include/tang.hpp File Reference	413
6.50.1 Detailed Description	414
6.51 include/tangBase.hpp File Reference	414
6.51.1 Detailed Description	414
6.52 include/tangScanner.hpp File Reference	415
6.52.1 Detailed Description	416
6.53 include/unescape.hpp File Reference	416
6.53.1 Detailed Description	417
6.54 include/unicodeString.hpp File Reference	417
6.54.1 Detailed Description	418
6.54.2 Function Documentation	418
6.54.2.1 htmlEscape()	418
6.54.2.2 htmlEscapeAscii()	418
6.54.2.3 unescape()	419
6.55 src/astNode.cpp File Reference	420
6.55.1 Detailed Description	420
6.56 src/astNodeArray.cpp File Reference	420
6.56.1 Detailed Description	421
6.57 src/astNodeAssign.cpp File Reference	421
6.57.1 Detailed Description	421

6.58 src/astNodeBinary.cpp File Reference	421
6.58.1 Detailed Description	422
6.59 src/astNodeBlock.cpp File Reference	422
6.59.1 Detailed Description	422
6.60 src/astNodeBoolean.cpp File Reference	423
6.60.1 Detailed Description	423
6.61 src/astNodeBreak.cpp File Reference	423
6.61.1 Detailed Description	424
6.62 src/astNodeCast.cpp File Reference	424
6.62.1 Detailed Description	425
6.63 src/astNodeContinue.cpp File Reference	425
6.63.1 Detailed Description	425
6.64 src/astNodeDoWhile.cpp File Reference	425
6.64.1 Detailed Description	426
6.65 src/astNodeFloat.cpp File Reference	426
6.65.1 Detailed Description	427
6.66 src/astNodeFor.cpp File Reference	427
6.66.1 Detailed Description	427
6.67 src/astNodeFunctionCall.cpp File Reference	427
6.67.1 Detailed Description	428
6.68 src/astNodeFunctionDeclaration.cpp File Reference	428
6.68.1 Detailed Description	429
6.69 src/astNodeIdentifier.cpp File Reference	429
6.69.1 Detailed Description	429
6.70 src/astNodeIfElse.cpp File Reference	429
6.70.1 Detailed Description	430
6.71 src/astNodeIndex.cpp File Reference	430
6.71.1 Detailed Description	431
6.72 src/astNodeInteger.cpp File Reference	431
6.72.1 Detailed Description	431
6.73 src/astNodeMap.cpp File Reference	431
6.73.1 Detailed Description	432
6.74 src/astNodePeriod.cpp File Reference	432
6.74.1 Detailed Description	433
6.75 src/astNodePrint.cpp File Reference	433
6.75.1 Detailed Description	433
6.76 src/astNodeRangedFor.cpp File Reference	433
6.76.1 Detailed Description	434
6.77 src/astNodeReturn.cpp File Reference	434
6.77.1 Detailed Description	434
6.78 src/astNodeSlice.cpp File Reference	435
6.78.1 Detailed Description	435

6.79 src/astNodeString.cpp File Reference	435
6.79.1 Detailed Description	436
6.80 src/astNodeTernary.cpp File Reference	436
6.80.1 Detailed Description	437
6.81 src/astNodeUnary.cpp File Reference	437
6.81.1 Detailed Description	437
6.82 src/astNodeWhile.cpp File Reference	437
6.82.1 Detailed Description	438
6.83 src/computedExpression.cpp File Reference	438
6.83.1 Detailed Description	439
6.84 src/computedExpressionArray.cpp File Reference	439
6.84.1 Detailed Description	439
6.85 src/computedExpressionBoolean.cpp File Reference	439
6.85.1 Detailed Description	440
6.86 src/computedExpressionCompiledFunction.cpp File Reference	440
6.86.1 Detailed Description	440
6.87 src/computedExpressionError.cpp File Reference	441
6.87.1 Detailed Description	441
6.88 src/computedExpressionFloat.cpp File Reference	441
6.88.1 Detailed Description	442
6.89 src/computedExpressionInteger.cpp File Reference	442
6.89.1 Detailed Description	442
6.90 src/computedExpressionIterator.cpp File Reference	443
6.90.1 Detailed Description	443
6.91 src/computedExpressionIteratorEnd.cpp File Reference	443
6.91.1 Detailed Description	444
6.92 src/computedExpressionMap.cpp File Reference	444
6.92.1 Detailed Description	444
6.93 src/computedExpressionNativeBoundFunction.cpp File Reference	445
6.93.1 Detailed Description	445
6.94 src/computedExpressionString.cpp File Reference	445
6.94.1 Detailed Description	446
6.95 src/error.cpp File Reference	446
6.95.1 Detailed Description	446
6.95.2 Function Documentation	446
6.95.2.1 operator<<()	446
6.96 src/garbageCollected.cpp File Reference	447
6.96.1 Function Documentation	447
6.96.1.1 operator<<()	447
6.97 src/program-dumpBytecode.cpp File Reference	448
6.97.1 Detailed Description	448
6.97.2 Macro Definition Documentation	448

6.97.2.1 DUMPPROGRAMCHECK	448
6.98 src/program-execute.cpp File Reference	449
6.98.1 Detailed Description	449
6.98.2 Macro Definition Documentation	449
6.98.2.1 EXECUTEPROGRAMCHECK	450
6.98.2.2 STACKCHECK	450
6.99 src/program.cpp File Reference	450
6.99.1 Detailed Description	451
6.100 src/tangBase.cpp File Reference	451
6.100.1 Detailed Description	451
6.101 src/unicodeString.cpp File Reference	452
6.101.1 Detailed Description	452
6.102 test/test.cpp File Reference	452
6.102.1 Detailed Description	453
6.103 test/testGarbageCollected.cpp File Reference	454
6.103.1 Detailed Description	454
6.104 test/testSingletonObjectPool.cpp File Reference	454
6.104.1 Detailed Description	455
6.105 test/testUnicodeString.cpp File Reference	455
6.105.1 Detailed Description	456
Index	457

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	20
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::AstNodeMap	95
Tang::AstNodePeriod	99
Tang::AstNodePrint	103
Tang::AstNodeRangedFor	108
Tang::AstNodeReturn	112
Tang::AstNodeSlice	116
Tang::AstNodeString	121
Tang::AstNodeTernary	126
Tang::AstNodeUnary	130
Tang::AstNodeWhile	136
Tang::ComputedExpression	140
Tang::ComputedExpressionArray	153
Tang::ComputedExpressionBoolean	169
Tang::ComputedExpressionCompiledFunction	182
Tang::ComputedExpressionError	195
Tang::ComputedExpressionFloat	210
Tang::ComputedExpressionInteger	224

Tang::ComputedExpressionIterator	239
Tang::ComputedExpressionIteratorEnd	253
Tang::ComputedExpressionMap	267
Tang::ComputedExpressionNativeBoundFunction	281
Tang::ComputedExpressionString	295
std::enable_shared_from_this	
Tang::TangBase	351
Tang::Error	311
Tang::GarbageCollected	313
Tang::location	334
Tang::position	336
Tang::Program	337
Tang::SingletonObjectPool< T >	348
TangHtmlEscapeAsciiFlexLexer	
Tang::HtmlEscapeAscii	332
TangHtmlEscapeFlexLexer	
Tang::HtmlEscape	330
TangTangFlexLexer	
Tang::TangScanner	353
TangUnescapeFlexLexer	
Tang::Unescape	356
Tang::UnicodeString	358

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	20
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::AstNodeMap	An <code>AstNode</code> that represents a map literal	95
Tang::AstNodePeriod	An <code>AstNode</code> that represents a member access (period) into an object	99
Tang::AstNodePrint	An <code>AstNode</code> that represents a print typeeration	103
Tang::AstNodeRangedFor	An <code>AstNode</code> that represents a ranged for() statement	108
Tang::AstNodeReturn	An <code>AstNode</code> that represents a <code>return</code> statement	112
Tang::AstNodeSlice	An <code>AstNode</code> that represents a ternary expression	116
Tang::AstNodeString	An <code>AstNode</code> that represents a string literal	121
Tang::AstNodeTernary	An <code>AstNode</code> that represents a ternary expression	126
Tang::AstNodeUnary	An <code>AstNode</code> that represents a unary negation	130
Tang::AstNodeWhile	An <code>AstNode</code> that represents a while statement	136
Tang::ComputedExpression	Represents the result of a computation that has been executed	140
Tang::ComputedExpressionArray	Represents an Array that is the result of a computation	153
Tang::ComputedExpressionBoolean	Represents an Boolean that is the result of a computation	169
Tang::ComputedExpressionCompiledFunction	Represents a Compiled Function declared in the script	182
Tang::ComputedExpressionError	Represents a Runtime <code>Error</code>	195
Tang::ComputedExpressionFloat	Represents a Float that is the result of a computation	210
Tang::ComputedExpressionInteger	Represents an Integer that is the result of a computation	224
Tang::ComputedExpressionIterator	Represents an Iterator that is the result of a computation	239
Tang::ComputedExpressionIteratorEnd	Represents that a collection has no more values through which to iterate	253
Tang::ComputedExpressionMap	Represents an Map that is the result of a computation	267
Tang::ComputedExpressionNativeBoundFunction	Represents a NativeBound Function declared in the script	281
Tang::ComputedExpressionString	Represents a String that is the result of a computation	295
Tang::Error	Used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error	311
Tang::GarbageCollected	A container that acts as a resource-counting garbage collector for the specified type	313
Tang::HtmlEscape	The Flex lexer class for the main Tang language	330
Tang::HtmlEscapeAscii	The Flex lexer class for the main Tang language	332
Tang::location	Two points in a source file	334
Tang::position	A point in a source file	336

Tang::Program	Represents a compiled script or template that may be executed	337
Tang::SingletonObjectPool< T >	A thread-safe, singleton object pool of the designated type	348
Tang::TangBase	The base class for the Tang programming language	351
Tang::TangScanner	The Flex lexer class for the main Tang language	353
Tang::Unescape	The Flex lexer class for the main Tang language	356
Tang::UnicodeString	Represents a UTF-8 encoded string that is Unicode-aware	358

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	363
include/ astNode.hpp	Declare the Tang::AstNode base class	365
include/ astNodeArray.hpp	Declare the Tang::AstNodeArray class	366
include/ astNodeAssign.hpp	Declare the Tang::AstNodeAssign class	367
include/ astNodeBinary.hpp	Declare the Tang::AstNodeBinary class	368
include/ astNodeBlock.hpp	Declare the Tang::AstNodeBlock class	369
include/ astNodeBoolean.hpp	Declare the Tang::AstNodeBoolean class	370
include/ astNodeBreak.hpp	Declare the Tang::AstNodeBreak class	371
include/ astNodeCast.hpp	Declare the Tang::AstNodeCast class	372
include/ astNodeContinue.hpp	Declare the Tang::AstNodeContinue class	373
include/ astNodeDoWhile.hpp	Declare the Tang::AstNodeDoWhile class	374
include/ astNodeFloat.hpp	Declare the Tang::AstNodeFloat class	375
include/ astNodeFor.hpp	Declare the Tang::AstNodeFor class	376
include/ astNodeFunctionCall.hpp	Declare the Tang::AstNodeFunctionCall class	377
include/ astNodeFunctionDeclaration.hpp	Declare the Tang::AstNodeFunctionDeclaration class	378
include/ astNodeIdentifier.hpp	Declare the Tang::AstNodeIdentifier class	379
include/ astNodeIfElse.hpp	Declare the Tang::AstNodeIfElse class	380
include/ astNodeIndex.hpp	Declare the Tang::AstNodeIndex class	381

include/astNodeInteger.hpp	Declare the <code>Tang::AstNodeInteger</code> class	382
include/astNodeMap.hpp	Declare the <code>Tang::AstNodeMap</code> class	383
include/astNodePeriod.hpp	Declare the <code>Tang::AstNodePeriod</code> class	384
include/astNodePrint.hpp	Declare the <code>Tang::AstNodePrint</code> class	385
include/astNodeRangedFor.hpp	Declare the <code>Tang::AstNodeRangedFor</code> class	386
include/astNodeReturn.hpp	Declare the <code>Tang::AstNodeReturn</code> class	387
include/astNodeSlice.hpp	Declare the <code>Tang::AstNodeSlice</code> class	388
include/astNodeString.hpp	Declare the <code>Tang::AstNodeString</code> class	389
include/astNodeTernary.hpp	Declare the <code>Tang::AstNodeTernary</code> class	390
include/astNodeUnary.hpp	Declare the <code>Tang::AstNodeUnary</code> class	391
include/astNodeWhile.hpp	Declare the <code>Tang::AstNodeWhile</code> class	392
include/computedExpression.hpp	Declare the <code>Tang::ComputedExpression</code> base class	393
include/computedExpressionArray.hpp	Declare the <code>Tang::ComputedExpressionArray</code> class	394
include/computedExpressionBoolean.hpp	Declare the <code>Tang::ComputedExpressionBoolean</code> class	395
include/computedExpressionCompiledFunction.hpp	Declare the <code>Tang::ComputedExpressionCompiledFunction</code> class	395
include/computedExpressionError.hpp	Declare the <code>Tang::ComputedExpressionError</code> class	396
include/computedExpressionFloat.hpp	Declare the <code>Tang::ComputedExpressionFloat</code> class	397
include/computedExpressionInteger.hpp	Declare the <code>Tang::ComputedExpressionInteger</code> class	398
include/computedExpressionIterator.hpp	Declare the <code>Tang::ComputedExpressionIterator</code> class	399
include/computedExpressionIteratorEnd.hpp	Declare the <code>Tang::ComputedExpressionIteratorEnd</code> class	400
include/computedExpressionMap.hpp	Declare the <code>Tang::ComputedExpressionMap</code> class	401
include/computedExpressionNativeBoundFunction.hpp	Declare the <code>Tang::ComputedExpressionNativeBoundFunction</code> class	402
include/computedExpressionString.hpp	Declare the <code>Tang::ComputedExpressionString</code> class	403
include/error.hpp	Declare the <code>Tang::Error</code> class used to describe syntax and runtime errors	404
include/garbageCollected.hpp	Declare the <code>Tang::GarbageCollected</code> class	405
include/htmlEscape.hpp	Declare the <code>Tang::HtmlEscape</code> used to tokenize a Tang script	406
include/htmlEscapeAscii.hpp	Declare the <code>Tang::HtmlEscapeAscii</code> used to tokenize a Tang script	407
include/macros.hpp	Contains generic macros	408
include/opcode.hpp	Declare the Opcodes used in the Bytecode representation of a program	409

include/program.hpp	Declare the <code>Tang::Program</code> class used to compile and execute source code	411
include/singletonObjectPool.hpp	Declare the <code>Tang::SingletonObjectPool</code> class	412
include/tang.hpp	Header file supplied for use by 3rd party code so that they can easily include all necessary headers	413
include/tangBase.hpp	Declare the <code>Tang::TangBase</code> class used to interact with Tang	414
include/tangScanner.hpp	Declare the <code>Tang::TangScanner</code> used to tokenize a Tang script	415
include/unescape.hpp	Declare the <code>Tang::Unescape</code> used to tokenize a Tang script	416
include/unicodeString.hpp	Contains the code to interface with the ICU library	417
src/astNode.cpp	Define the <code>Tang::AstNode</code> class	420
src/astNodeArray.cpp	Define the <code>Tang::AstNodeArray</code> class	420
src/astNodeAssign.cpp	Define the <code>Tang::AstNodeAssign</code> class	421
src/astNodeBinary.cpp	Define the <code>Tang::AstNodeBinary</code> class	421
src/astNodeBlock.cpp	Define the <code>Tang::AstNodeBlock</code> class	422
src/astNodeBoolean.cpp	Define the <code>Tang::AstNodeBoolean</code> class	423
src/astNodeBreak.cpp	Define the <code>Tang::AstNodeBreak</code> class	423
src/astNodeCast.cpp	Define the <code>Tang::AstNodeCast</code> class	424
src/astNodeContinue.cpp	Define the <code>Tang::AstNodeContinue</code> class	425
src/astNodeDoWhile.cpp	Define the <code>Tang::AstNodeDoWhile</code> class	425
src/astNodeFloat.cpp	Define the <code>Tang::AstNodeFloat</code> class	426
src/astNodeFor.cpp	Define the <code>Tang::AstNodeFor</code> class	427
src/astNodeFunctionCall.cpp	Define the <code>Tang::AstNodeFunctionCall</code> class	427
src/astNodeFunctionDeclaration.cpp	Define the <code>Tang::AstNodeFunctionDeclaration</code> class	428
src/astNodeIdentifier.cpp	Define the <code>Tang::AstNodeIdentifier</code> class	429
src/astNodeIfElse.cpp	Define the <code>Tang::AstNodeIfElse</code> class	429
src/astNodeIndex.cpp	Define the <code>Tang::AstNodeIndex</code> class	430
src/astNodeInteger.cpp	Define the <code>Tang::AstNodeInteger</code> class	431
src/astNodeMap.cpp	Define the <code>Tang::AstNodeMap</code> class	431
src/astNodePeriod.cpp	Define the <code>Tang::AstNodePeriod</code> class	432
src/astNodePrint.cpp	Define the <code>Tang::AstNodePrint</code> class	433

src/astNodeRangedFor.cpp	Define the <code>Tang::AstNodeRangedFor</code> class	433
src/astNodeReturn.cpp	Define the <code>Tang::AstNodeReturn</code> class	434
src/astNodeSlice.cpp	Define the <code>Tang::AstNodeSlice</code> class	435
src/astNodeString.cpp	Define the <code>Tang::AstNodeString</code> class	435
src/astNodeTernary.cpp	Define the <code>Tang::AstNodeTernary</code> class	436
src/astNodeUnary.cpp	Define the <code>Tang::AstNodeUnary</code> class	437
src/astNodeWhile.cpp	Define the <code>Tang::AstNodeWhile</code> class	437
src/computedExpression.cpp	Define the <code>Tang::ComputedExpression</code> class	438
src/computedExpressionArray.cpp	Define the <code>Tang::ComputedExpressionArray</code> class	439
src/computedExpressionBoolean.cpp	Define the <code>Tang::ComputedExpressionBoolean</code> class	439
src/computedExpressionCompiledFunction.cpp	Define the <code>Tang::ComputedExpressionCompiledFunction</code> class	440
src/computedExpressionError.cpp	Define the <code>Tang::ComputedExpressionError</code> class	441
src/computedExpressionFloat.cpp	Define the <code>Tang::ComputedExpressionFloat</code> class	441
src/computedExpressionInteger.cpp	Define the <code>Tang::ComputedExpressionInteger</code> class	442
src/computedExpressionIterator.cpp	Define the <code>Tang::ComputedExpressionIterator</code> class	443
src/computedExpressionIteratorEnd.cpp	Define the <code>Tang::ComputedExpressionIteratorEnd</code> class	443
src/computedExpressionMap.cpp	Define the <code>Tang::ComputedExpressionMap</code> class	444
src/computedExpressionNativeBoundFunction.cpp	Define the <code>Tang::ComputedExpressionNativeBoundFunction</code> class	445
src/computedExpressionString.cpp	Define the <code>Tang::ComputedExpressionString</code> class	445
src/error.cpp	Define the <code>Tang::Error</code> class	446
src/garbageCollected.cpp		447
src/program-dumpBytecode.cpp	Define the <code>Tang::Program::dumpBytecode</code> method	448
src/program-execute.cpp	Define the <code>Tang::Program::execute</code> method	449
src/program.cpp	Define the <code>Tang::Program</code> class	450
src/tangBase.cpp	Define the <code>Tang::TangBase</code> class	451
src/unicodeString.cpp	Contains the function declarations for the <code>Tang::UnicodeString</code> class and the interface to ICU	452
test/test.cpp	Test the general language behaviors	452
test/testGarbageCollected.cpp	Test the generic behavior of the <code>Tang::GarbageCollected</code> class	454
test/testSingletonObjectPool.cpp	Test the generic behavior of the <code>Tang::SingletonObjectPool</code> class	454

test/testUnicodeString.cpp	Contains tests for the Tang::UnicodeString class	455
----------------------------	--	-----

Chapter 5

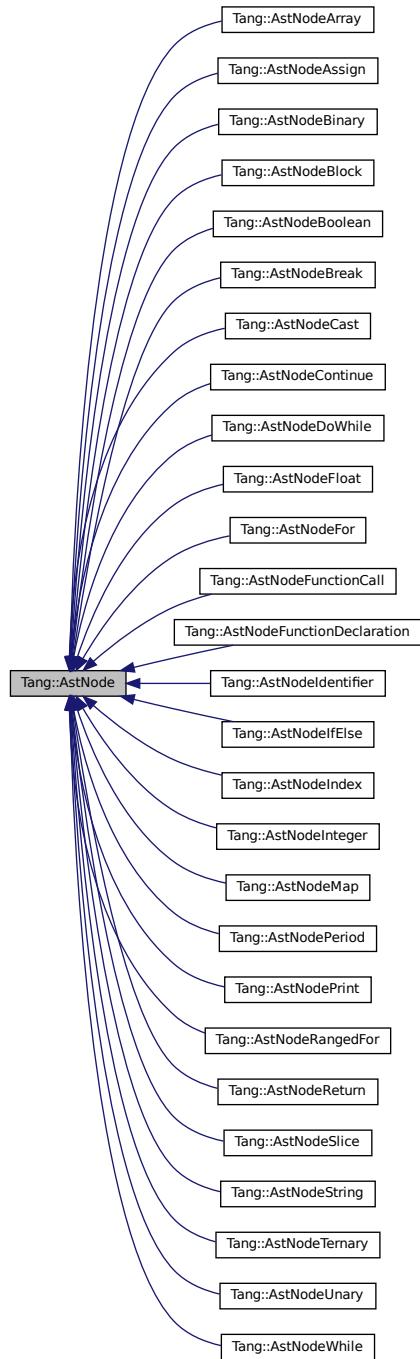
Class Documentation

5.1 Tang::AstNode Class Reference

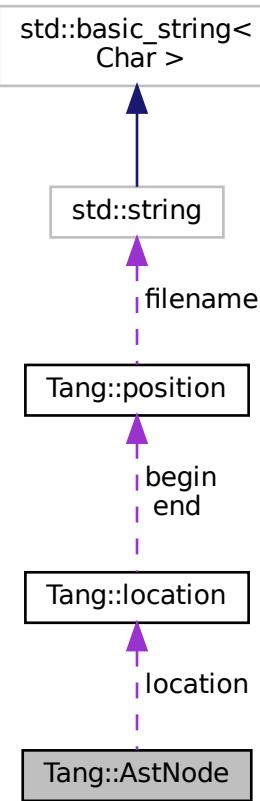
Base class for representing nodes of an Abstract Syntax Tree (AST).

```
#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::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#), [Tang::AstNodeMap](#), [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::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#),

`Tang::AstNodeMap`, `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

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

Returns

The value as a string.

Reimplemented in `Tang::AstNodeWhile`, `Tang::AstNodeUnary`, `Tang::AstNodeTernary`, `Tang::AstNodeString`, `Tang::AstNodeSlice`, `Tang::AstNodeReturn`, `Tang::AstNodeRangedFor`, `Tang::AstNodePrint`, `Tang::AstNodePeriod`, `Tang::AstNodeMap`, `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:

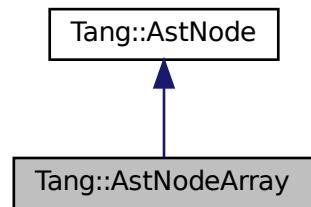
- `include/astNode.hpp`
- `src/astNode.cpp`

5.2 `Tang::AstNodeArray` Class Reference

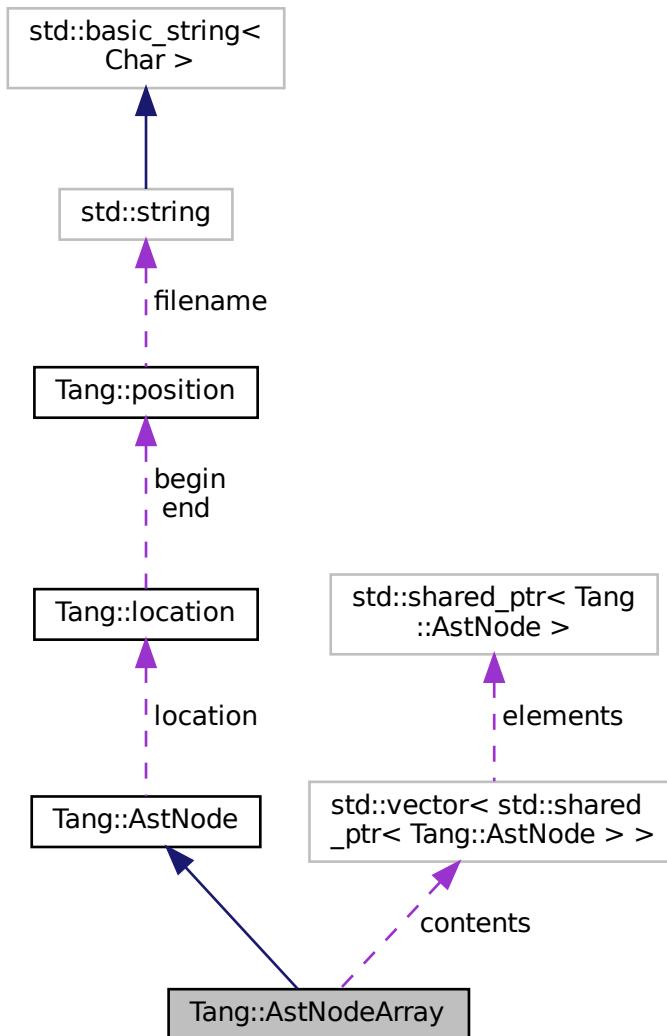
An `AstNode` that represents an array literal.

```
#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]
```

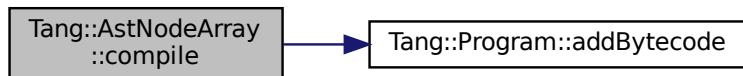
Compile the ast of the provided [Tang::Program](#).

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:

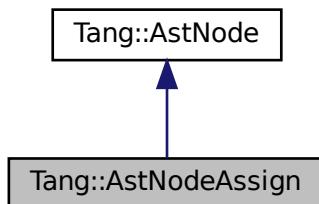
- [include/astNodeArray.hpp](#)
- [src/astNodeArray.cpp](#)

5.3 Tang::AstNodeAssign Class Reference

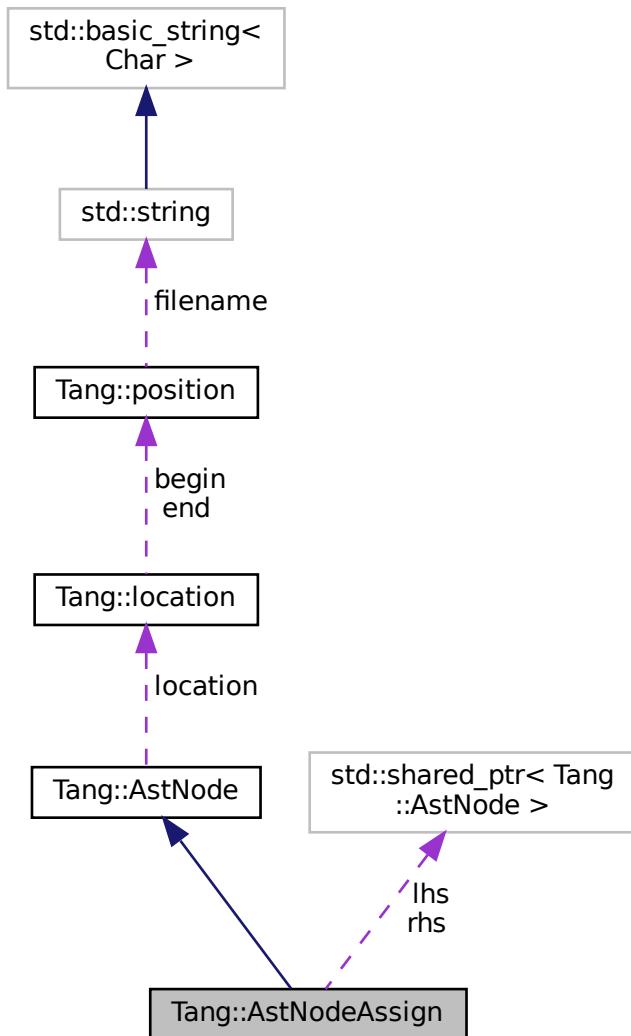
An [AstNode](#) that represents a binary expression.

```
#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]
```

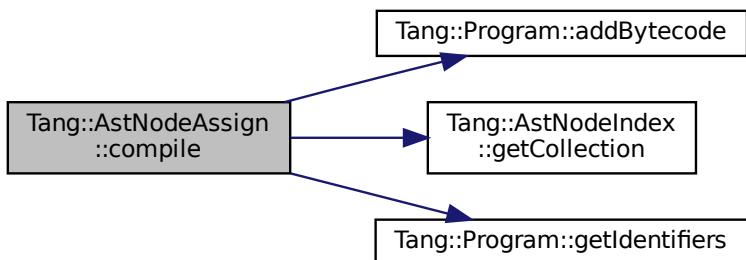
Compile the ast of the provided [Tang::Program](#).

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:

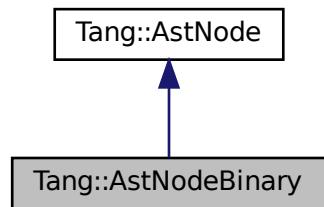
- [include/astNodeAssign.hpp](#)
- [src/astNodeAssign.cpp](#)

5.4 Tang::AstNodeBinary Class Reference

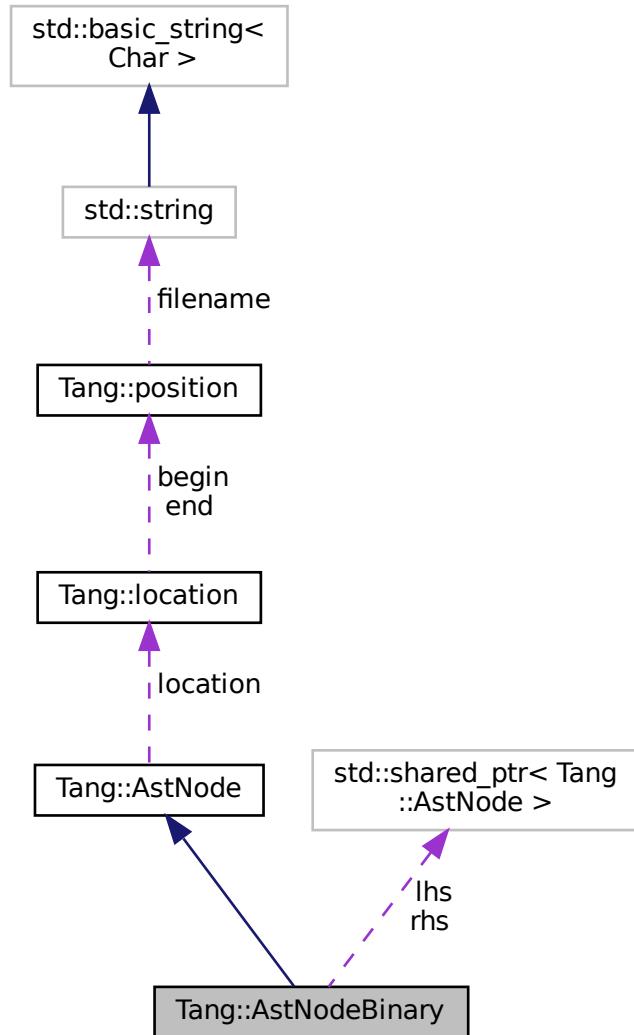
An [AstNode](#) that represents a binary expression.

```
#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]
```

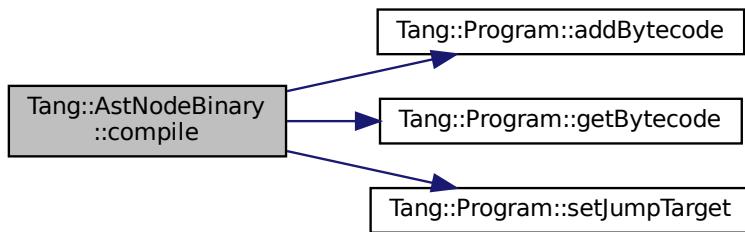
Compile the ast of the provided [Tang::Program](#).

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:

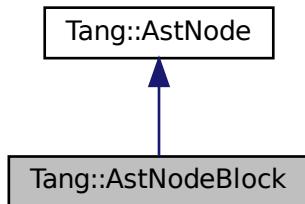
- [include/astNodeBinary.hpp](#)
- [src/astNodeBinary.cpp](#)

5.5 Tang::AstNodeBlock Class Reference

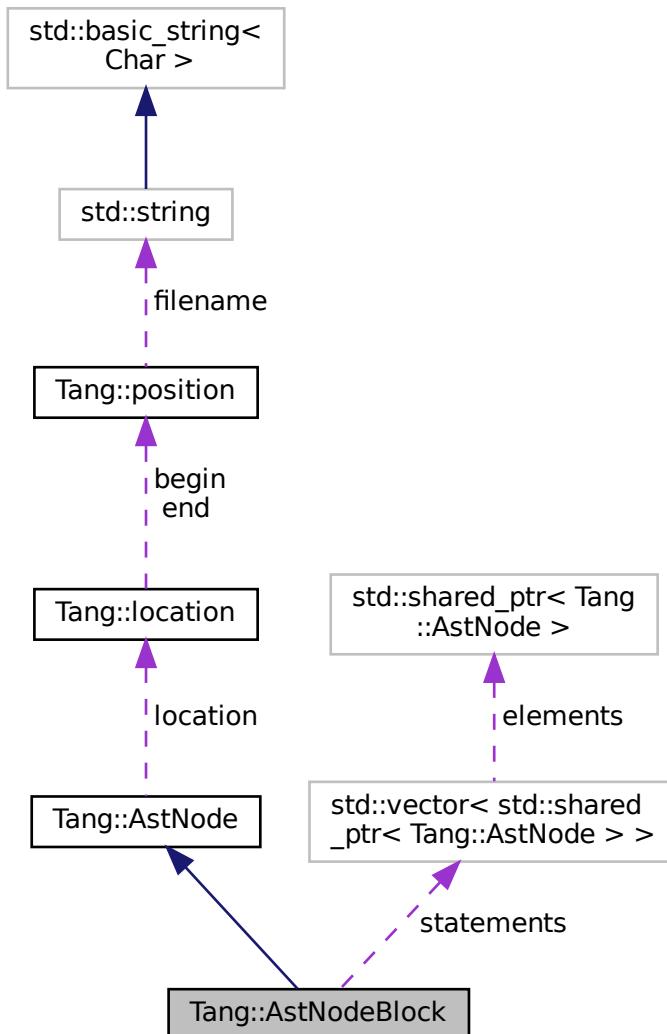
An [AstNode](#) that represents a code block.

```
#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]
```

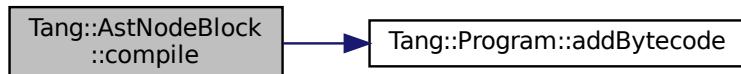
Compile the ast of the provided [Tang::Program](#).

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:

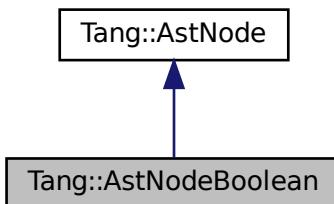
- [include/astNodeBlock.hpp](#)
- [src/astNodeBlock.cpp](#)

5.6 Tang::AstNodeBoolean Class Reference

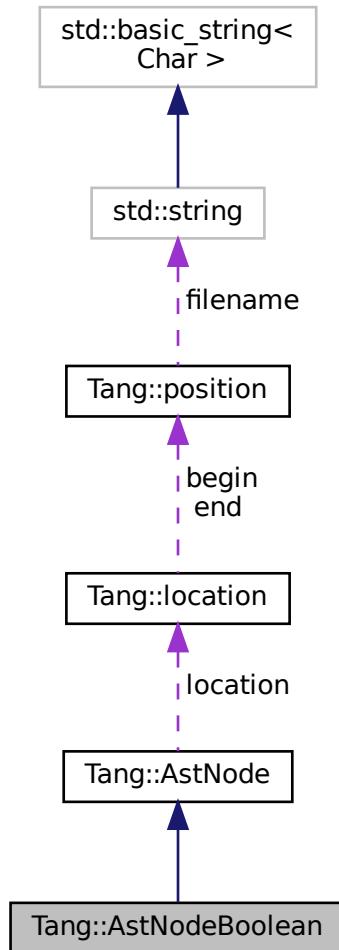
An [AstNode](#) that represents a boolean literal.

```
#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]
```

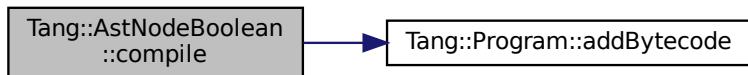
Compile the ast of the provided [Tang::Program](#).

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::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#), [Tang::AstNodeMap](#), [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 (
    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:

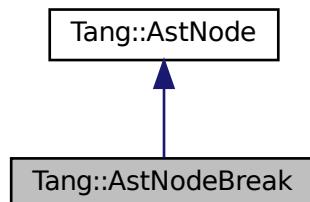
- [include/astNodeBoolean.hpp](#)
- [src/astNodeBoolean.cpp](#)

5.7 Tang::AstNodeBreak Class Reference

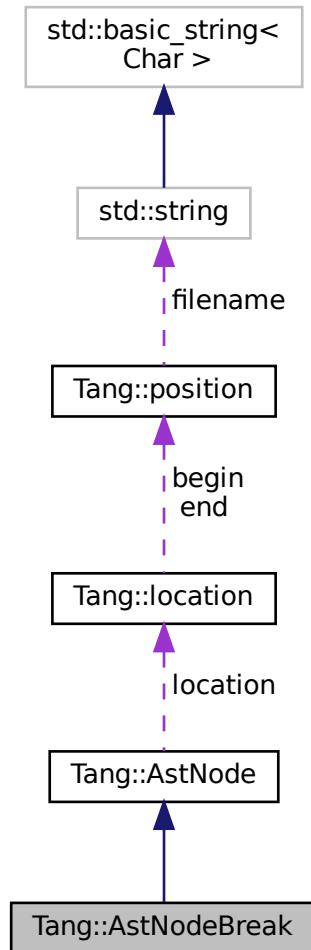
An [AstNode](#) that represents a `break` statement.

```
#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]
```

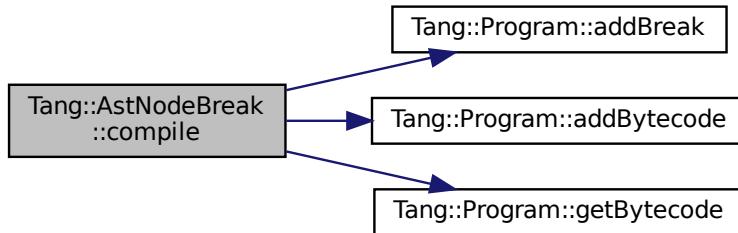
Compile the ast of the provided [Tang::Program](#).

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::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#), [Tang::AstNodeMap](#), [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:

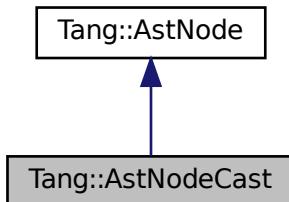
- [include/astNodeBreak.hpp](#)
- [src/astNodeBreak.cpp](#)

5.8 Tang::AstNodeCast Class Reference

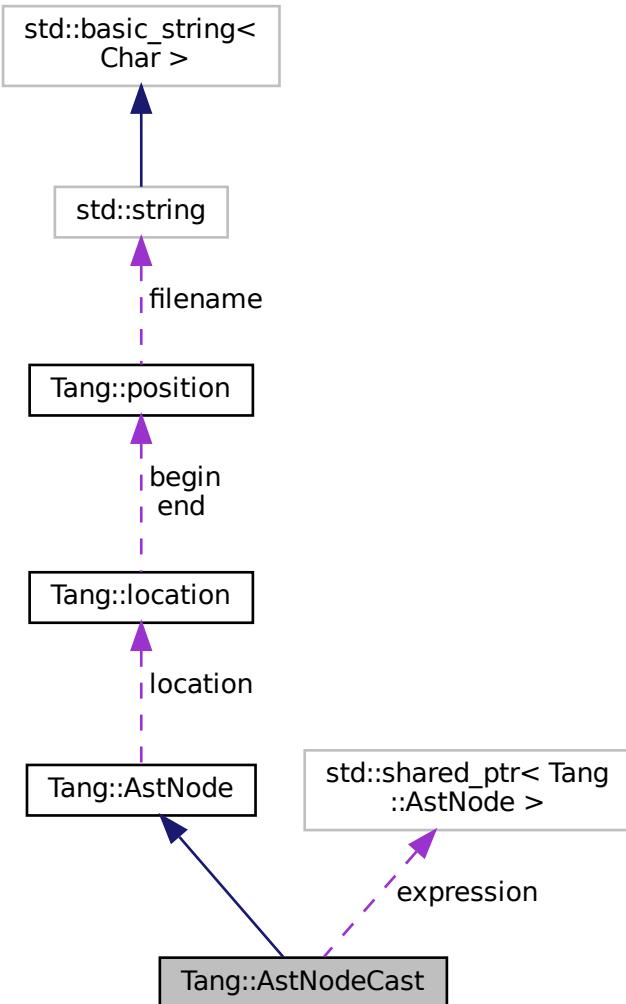
An [AstNode](#) that represents a typecast of an expression.

```
#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, 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.
- `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,
    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]
```

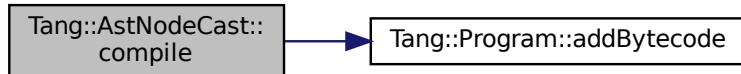
Compile the ast of the provided [Tang::Program](#).

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:

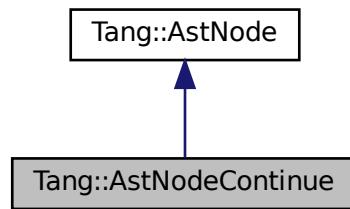
- [include/astNodeCast.hpp](#)
- [src/astNodeCast.cpp](#)

5.9 Tang::AstNodeContinue Class Reference

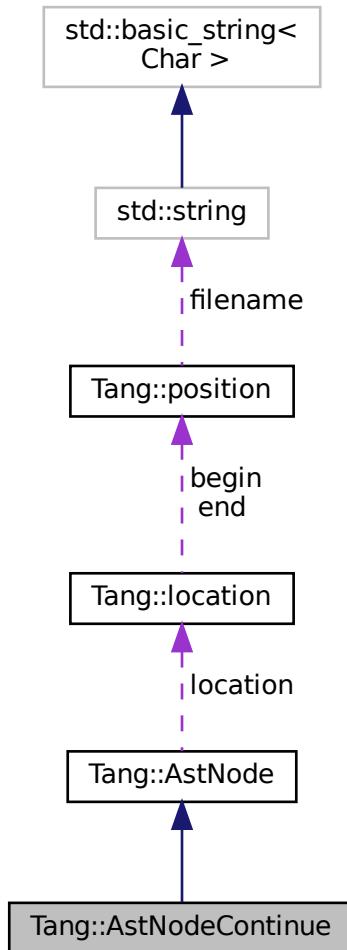
An [AstNode](#) that represents a `continue` statement.

```
#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]
```

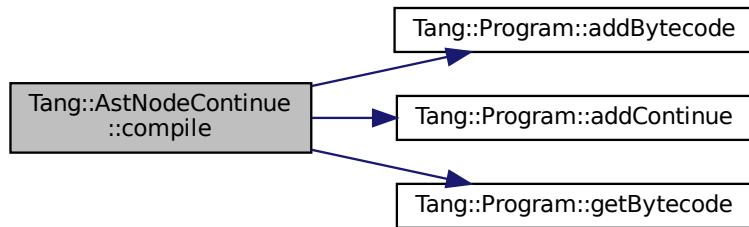
Compile the ast of the provided [Tang::Program](#).

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::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#), [Tang::AstNodeMap](#), [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](#).

5.9.4.3 `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:

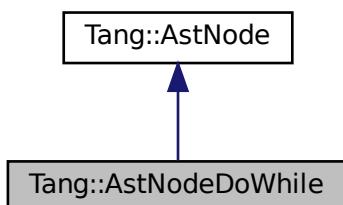
- [include/astNodeContinue.hpp](#)
- [src/astNodeContinue.cpp](#)

5.10 `Tang::AstNodeDoWhile` Class Reference

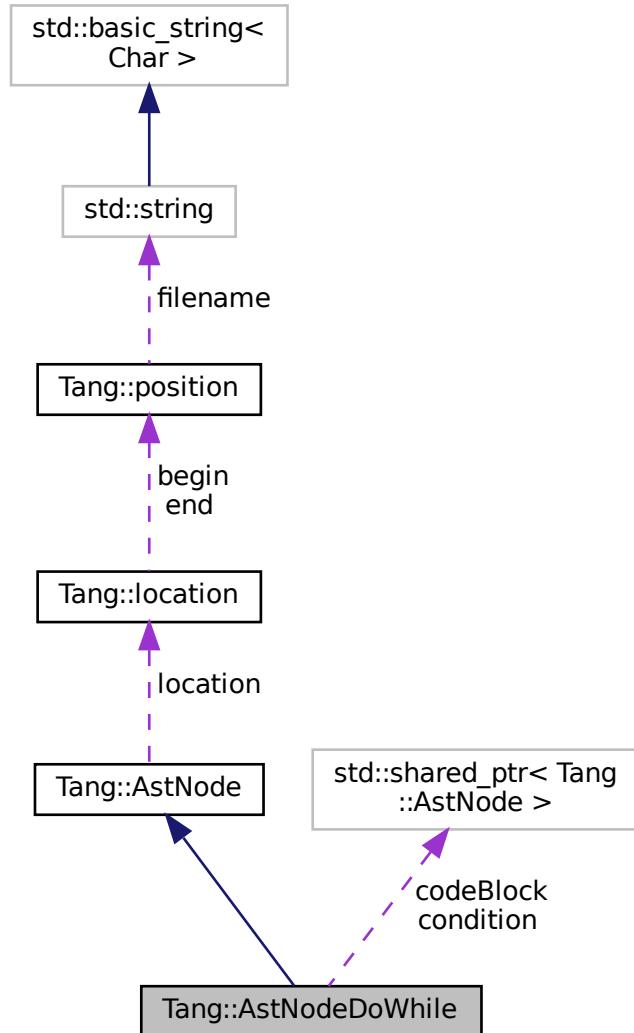
An [AstNode](#) that represents a do..while statement.

```
#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 (shared_ptr< AstNode > condition, 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

- `shared_ptr< AstNode > condition`
The expression which determines whether or not the code block will continue to be executed.
- `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 (
    shared_ptr< AstNode > condition,
```

```
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]
```

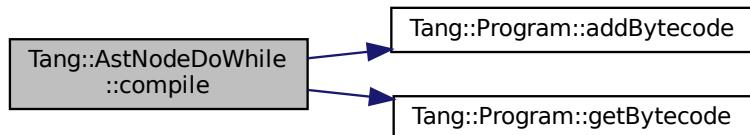
Compile the ast of the provided [Tang::Program](#).

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:

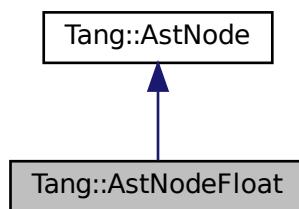
- `include/astNodeDoWhile.hpp`
- `src/astNodeDoWhile.cpp`

5.11 Tang::AstNodeFloat Class Reference

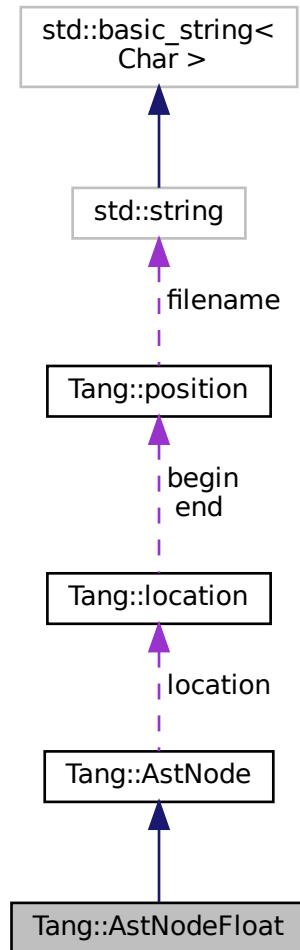
An [AstNode](#) that represents an float literal.

```
#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]
```

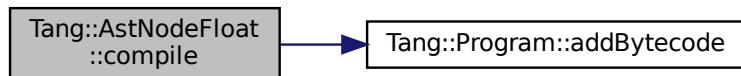
Compile the ast of the provided [Tang::Program](#).

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::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#), [Tang::AstNodeMap](#), [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:

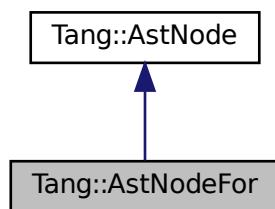
- [include/astNodeFloat.hpp](#)
- [src/astNodeFloat.cpp](#)

5.12 Tang::AstNodeFor Class Reference

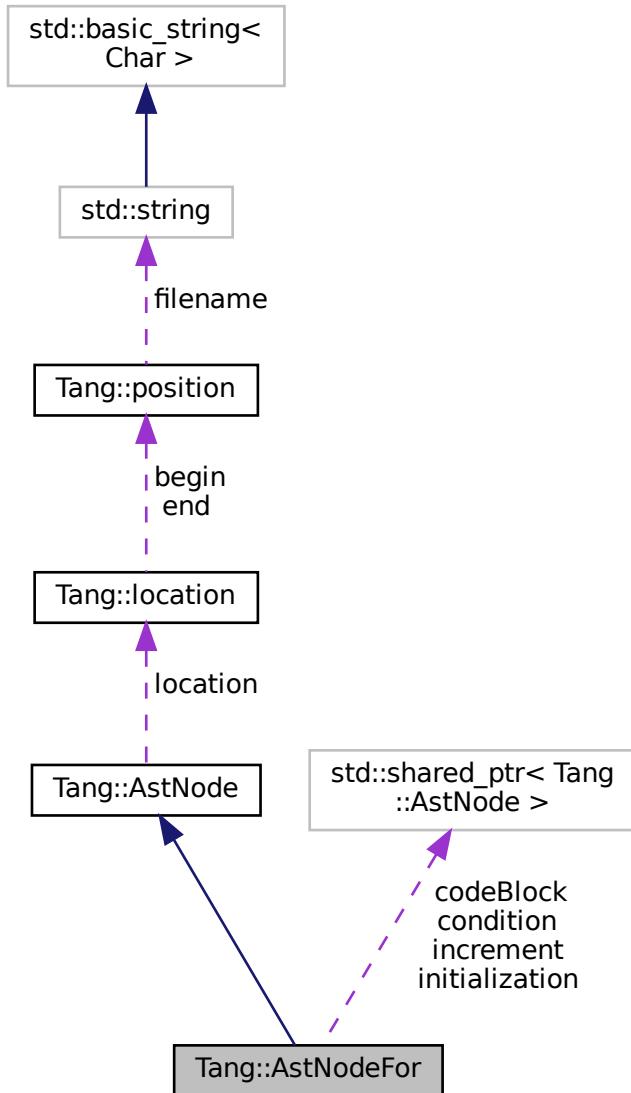
An [AstNode](#) that represents an if() statement.

```
#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** (`shared_ptr<AstNode> initialization, shared_ptr<AstNode> condition, shared_ptr<AstNode> increment, 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

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

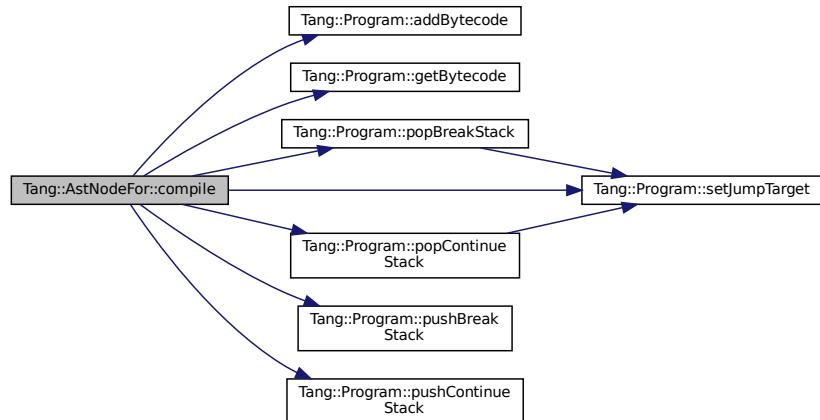
Compile the ast of the provided [Tang::Program](#).

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:

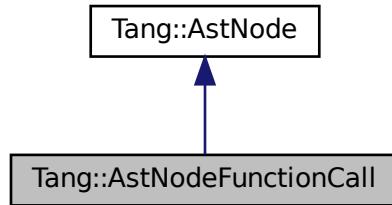
- [include/astNodeFor.hpp](#)
- [src/astNodeFor.cpp](#)

5.13 Tang::AstNodeFunctionCall Class Reference

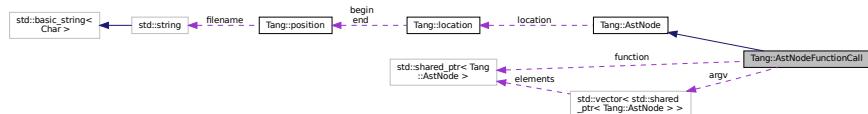
An [AstNode](#) that represents a function call.

```
#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:

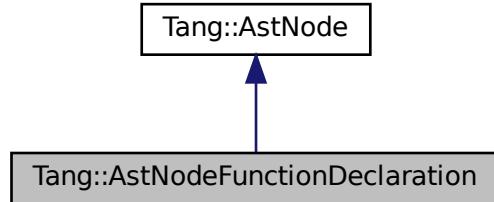
- [include/astNodeFunctionCall.hpp](#)
- [src/astNodeFunctionCall.cpp](#)

5.14 Tang::AstNodeFunctionDeclaration Class Reference

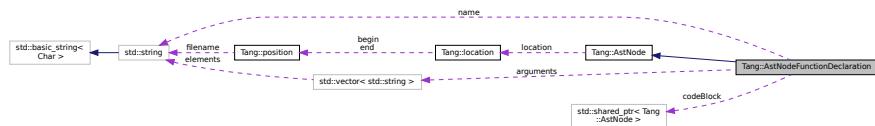
An [AstNode](#) that represents a function declaration.

```
#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](#), 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.
- 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,
    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]
```

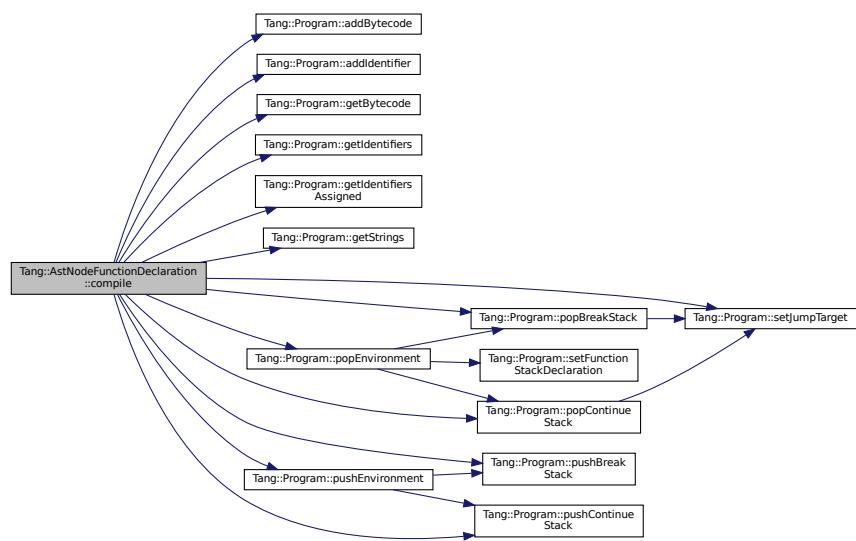
Compile the ast of the provided [Tang::Program](#).

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:

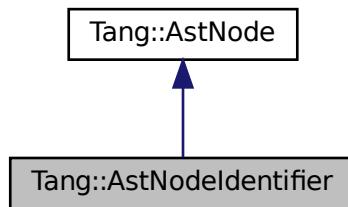
- [include/astNodeFunctionDeclaration.hpp](#)
- [src/astNodeFunctionDeclaration.cpp](#)

5.15 Tang::AstNodeIdentifier Class Reference

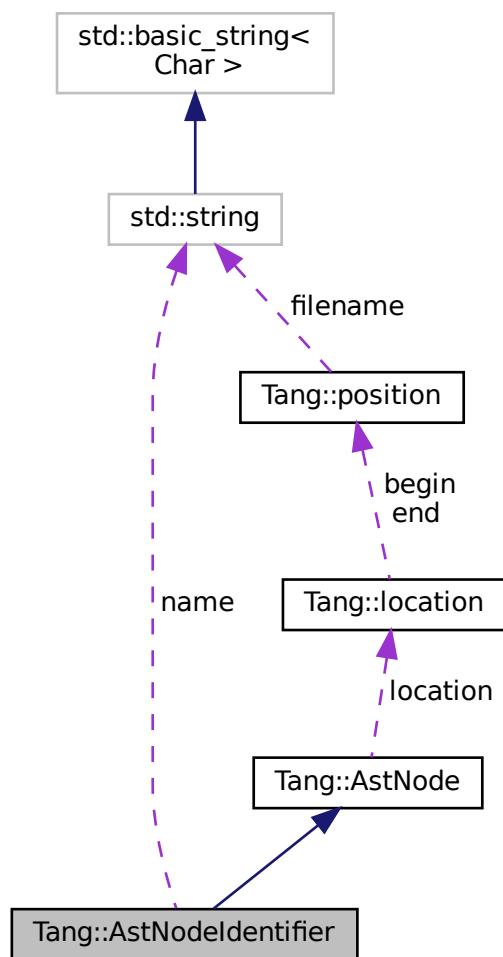
An [AstNode](#) that represents an identifier.

```
#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]
```

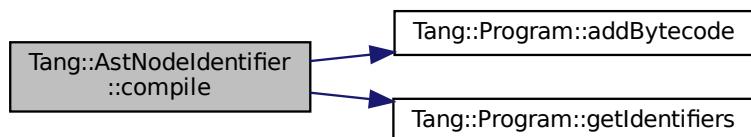
Compile the ast of the provided [Tang::Program](#).

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]
```

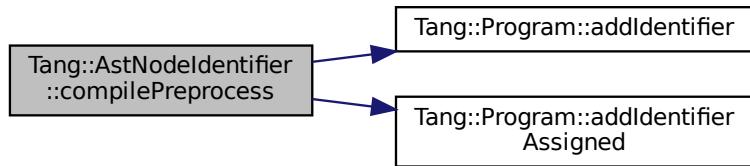
Run any preprocess analysis needed before compilation.

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:

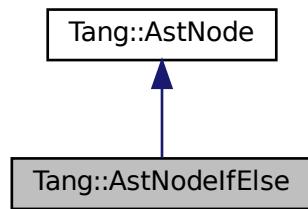
- [include/astNodeIdentifier.hpp](#)
- [src/astNodeIdentifier.cpp](#)

5.16 Tang::AstNodeIfElse Class Reference

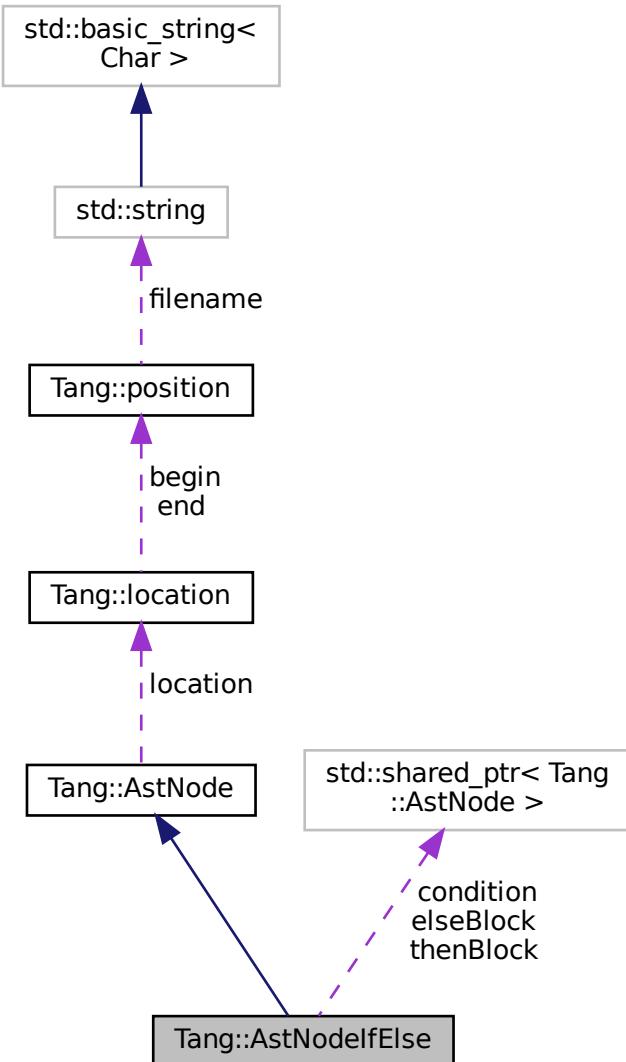
An [AstNode](#) that represents an if..else statement.

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

The constructor.

- `AstNodeIfElse` (`shared_ptr< AstNode > condition, 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

- `shared_ptr< AstNode > condition`
The expression which determines whether the thenBlock or elseBlock is executed.
- `shared_ptr< AstNode > thenBlock`
The statement executed when the condition is true.
- `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 (
    shared_ptr< AstNode > condition,
    shared_ptr< AstNode > thenBlock,
    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 (
    shared_ptr< AstNode > condition,
    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]
```

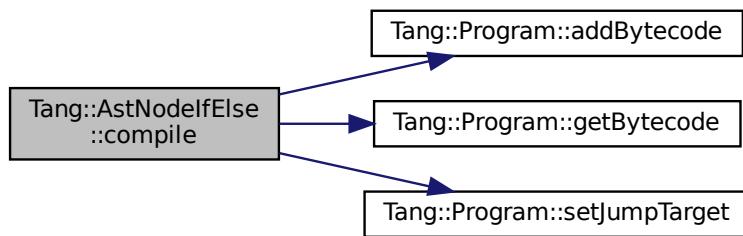
Compile the ast of the provided `Tang::Program`.

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:

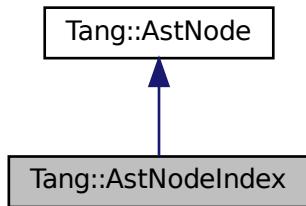
- [include/astNodeIfElse.hpp](#)
- [src/astNodeIfElse.cpp](#)

5.17 Tang::AstNodeIndex Class Reference

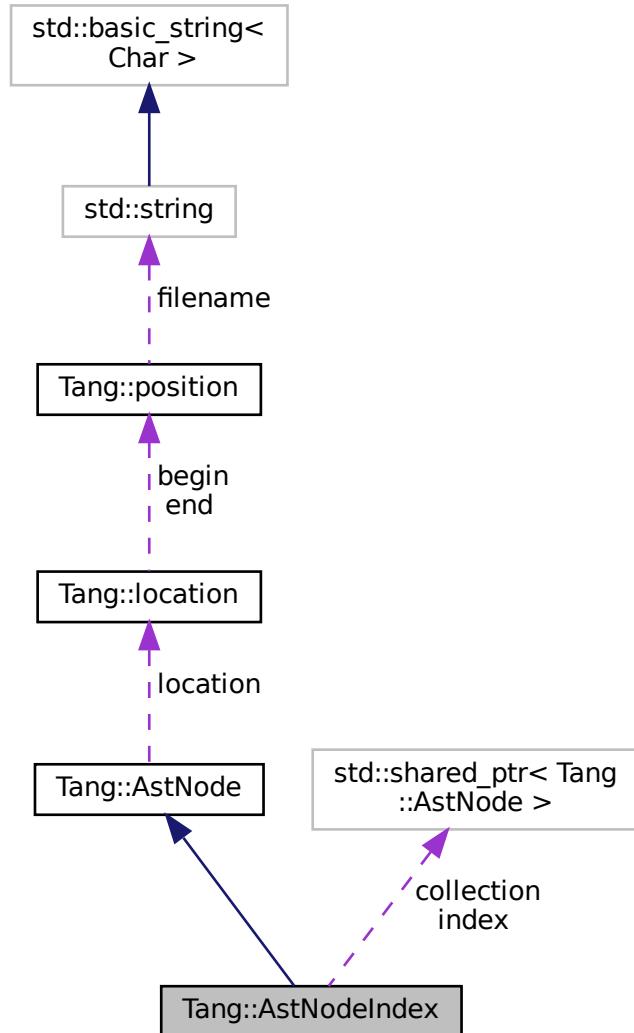
An [AstNode](#) that represents an index into a collection.

```
#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]
```

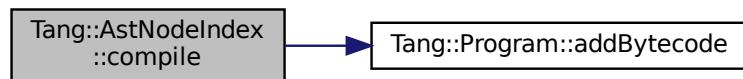
Compile the ast of the provided [Tang::Program](#).

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:

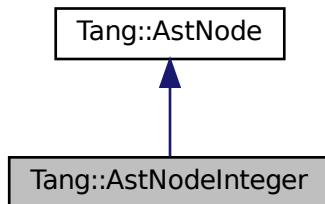
- [include/astNodeIndex.hpp](#)
- [src/astNodeIndex.cpp](#)

5.18 Tang::AstNodeInteger Class Reference

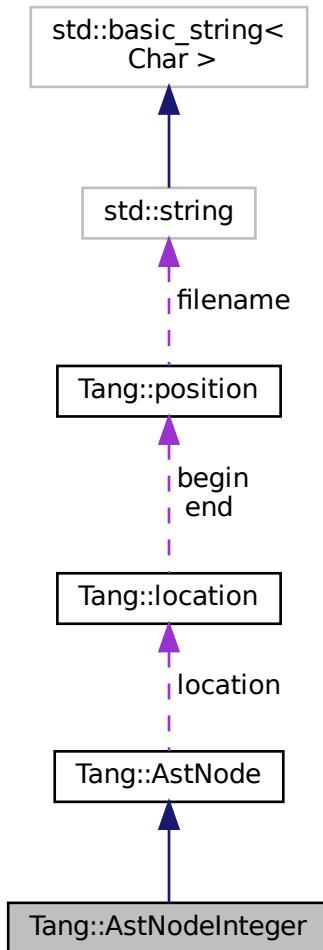
An [AstNode](#) that represents an integer literal.

```
#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]
```

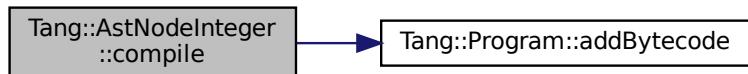
Compile the ast of the provided [Tang::Program](#).

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::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeSlice](#), [Tang::AstNodeReturn](#), [Tang::AstNodeRangedFor](#), [Tang::AstNodePrint](#), [Tang::AstNodePeriod](#), [Tang::AstNodeMap](#), [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:

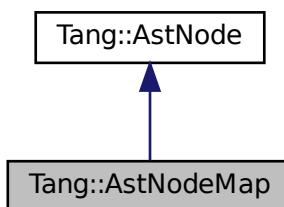
- [include/astNodeInteger.hpp](#)
- [src/astNodeInteger.cpp](#)

5.19 Tang::AstNodeMap Class Reference

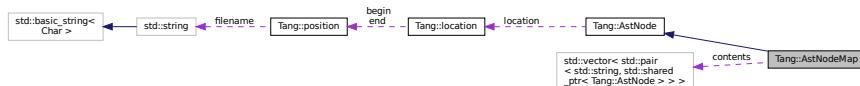
An [AstNode](#) that represents a map literal.

```
#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.

5.19.1 Detailed Description

An `AstNode` that represents a map literal.

Keys can only be strings.

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	AstNode is part of an assignment expression.

5.19.3 Constructor & Destructor Documentation**5.19.3.1 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.

5.19.4 Member Function Documentation**5.19.4.1 compile()**

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

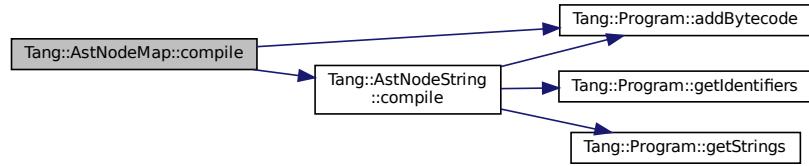
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.19.4.2 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:



5.19.4.3 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:

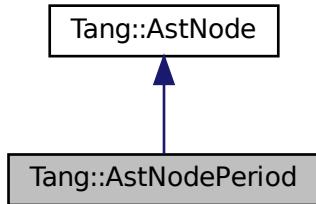
- [include/astNodeMap.hpp](#)
- [src/astNodeMap.cpp](#)

5.20 Tang::AstNodePeriod Class Reference

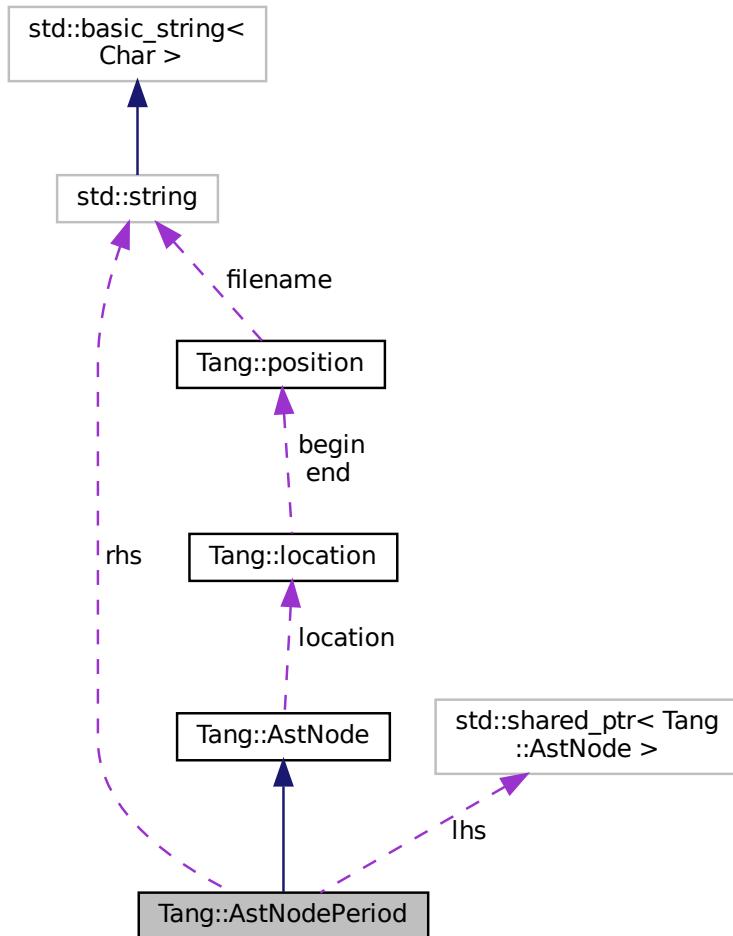
An [AstNode](#) that represents a member access (period) into an object.

```
#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.

5.20.1 Detailed Description

An `AstNode` that represents a member access (period) into an object.

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	<code>AstNode</code> is part of an assignment expression.

5.20.3 Constructor & Destructor Documentation

5.20.3.1 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.

5.20.4 Member Function Documentation

5.20.4.1 `compile()`

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

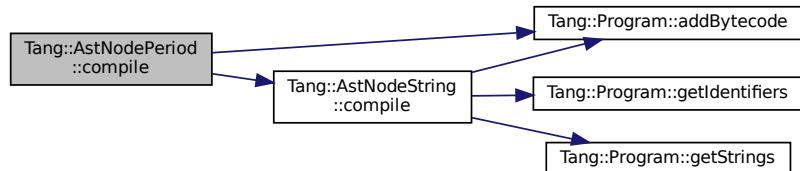
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:

5.20.4.2 `compilePreprocess()`

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

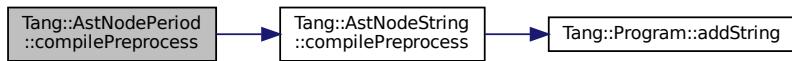
Run any preprocess analysis needed before compilation.

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:



5.20.4.3 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:

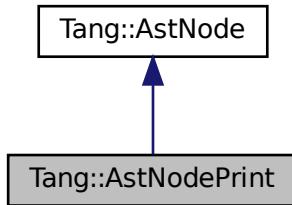
- [include/astNodePeriod.hpp](#)
- [src/astNodePeriod.cpp](#)

5.21 Tang::AstNodePrint Class Reference

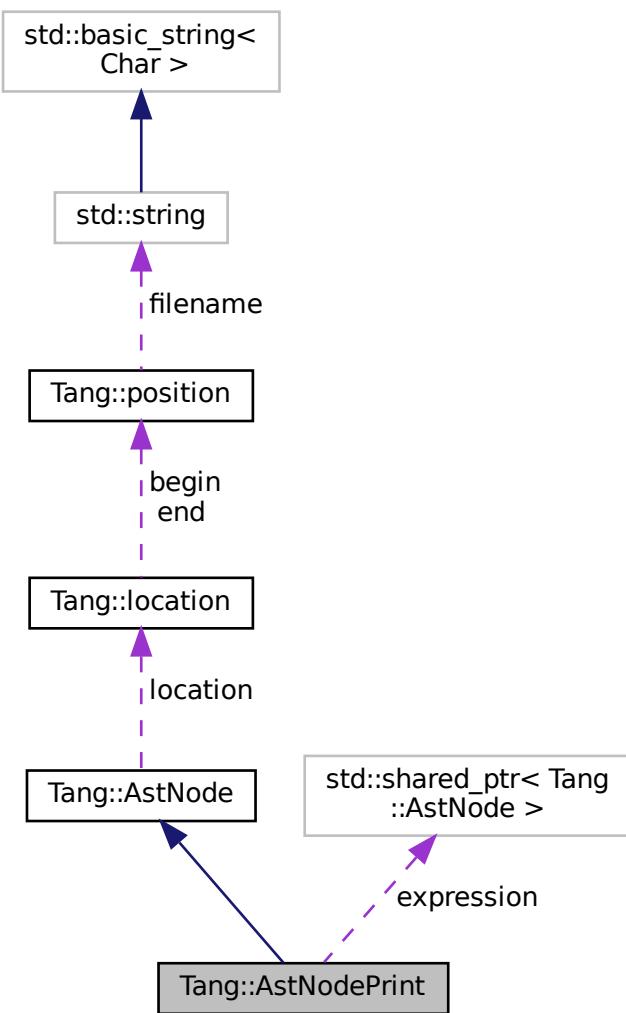
An [AstNode](#) that represents a print typeeration.

```
#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, 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.
- `shared_ptr< AstNode > expression`
The expression to be printed.

5.21.1 Detailed Description

An `AstNode` that represents a print typeeration.

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.

5.21.2.2 Type

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

The type of print() requested.

Enumerator

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

5.21.3 Constructor & Destructor Documentation**5.21.3.1 [AstNodePrint\(\)](#)**

```
AstNodePrint::AstNodePrint (
    Type type,
    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.

5.21.4 Member Function Documentation**5.21.4.1 [compile\(\)](#)**

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

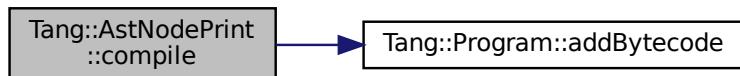
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The Tang::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 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](#).

5.21.4.3 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:

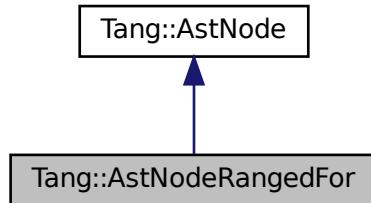
- [include/astNodePrint.hpp](#)
- [src/astNodePrint.cpp](#)

5.22 Tang::AstNodeRangedFor Class Reference

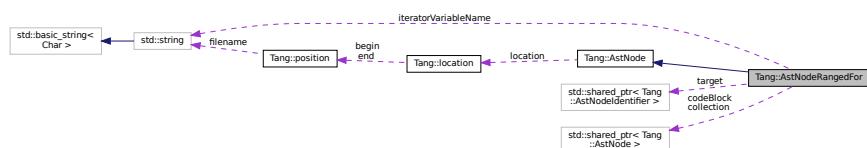
An [AstNode](#) that represents a ranged for() statement.

```
#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 (shared_ptr< AstNodelIdentifier > target, shared_ptr< AstNode > collection, 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

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

5.22.1 Detailed Description

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

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	<code>AstNode</code> is part of an assignment expression.

5.22.3 Constructor & Destructor Documentation

5.22.3.1 AstNodeRangedFor()

```
AstNodeRangedFor::AstNodeRangedFor (
    shared_ptr< AstNodeIdentifier > target,
    shared_ptr< AstNode > collection,
    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.

5.22.4 Member Function Documentation

5.22.4.1 compile()

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

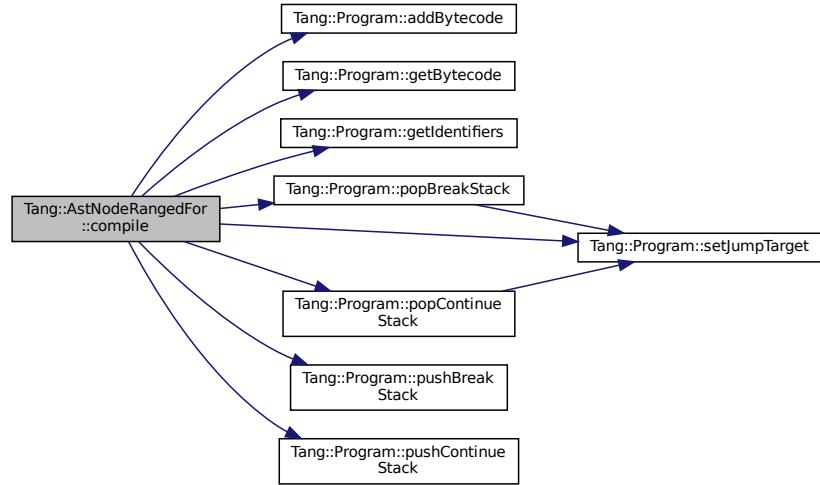
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.22.4.2 compilePreprocess()

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

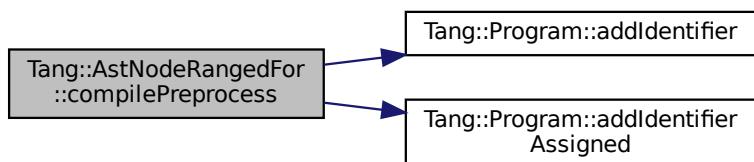
Run any preprocess analysis needed before compilation.

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:



5.22.4.3 `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:

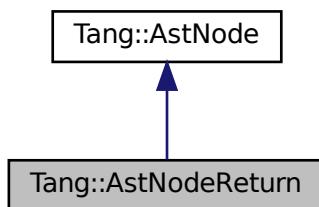
- [include/astNodeRangedFor.hpp](#)
- [src/astNodeRangedFor.cpp](#)

5.23 Tang::AstNodeReturn Class Reference

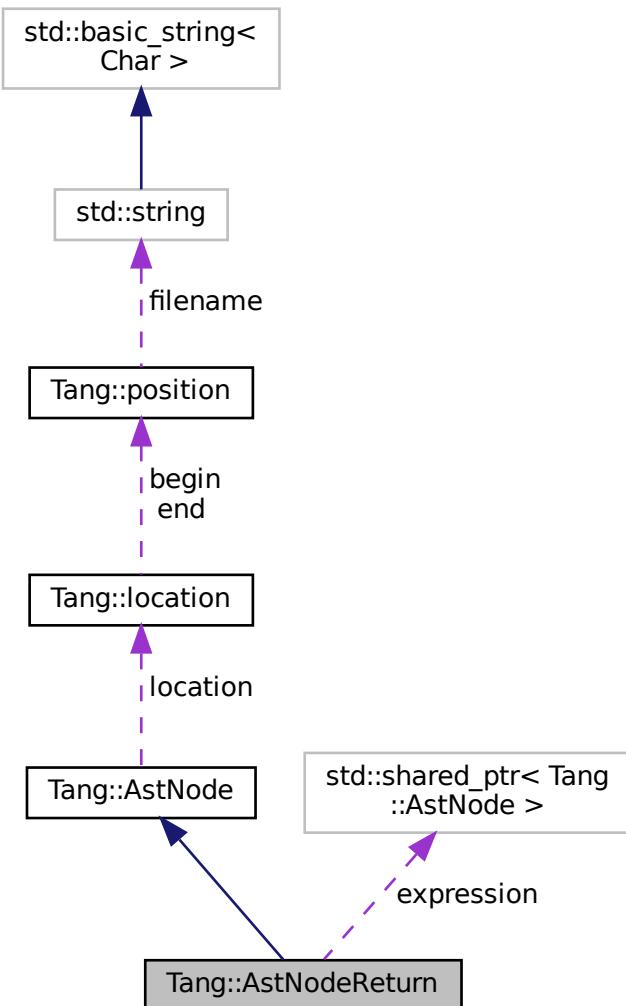
An [AstNode](#) that represents a `return` statement.

```
#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** (`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

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

5.23.1 Detailed Description

An `AstNode` that represents a `return` 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 AstNodeReturn()

```
AstNodeReturn::AstNodeReturn (
    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.

5.23.4 Member Function Documentation

5.23.4.1 compile()

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

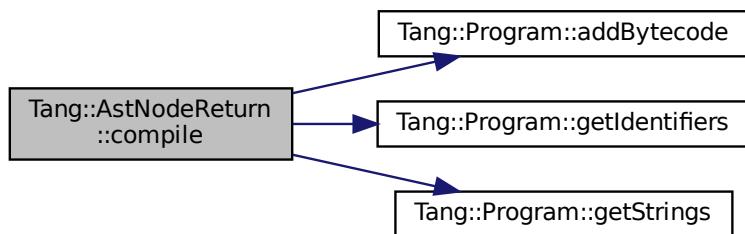
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.23.4.2 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](#).

5.23.4.3 `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:

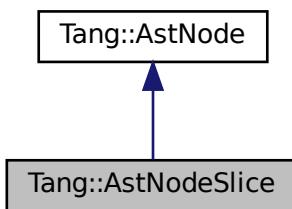
- [include/astNodeReturn.hpp](#)
- [src/astNodeReturn.cpp](#)

5.24 Tang::AstNodeSlice Class Reference

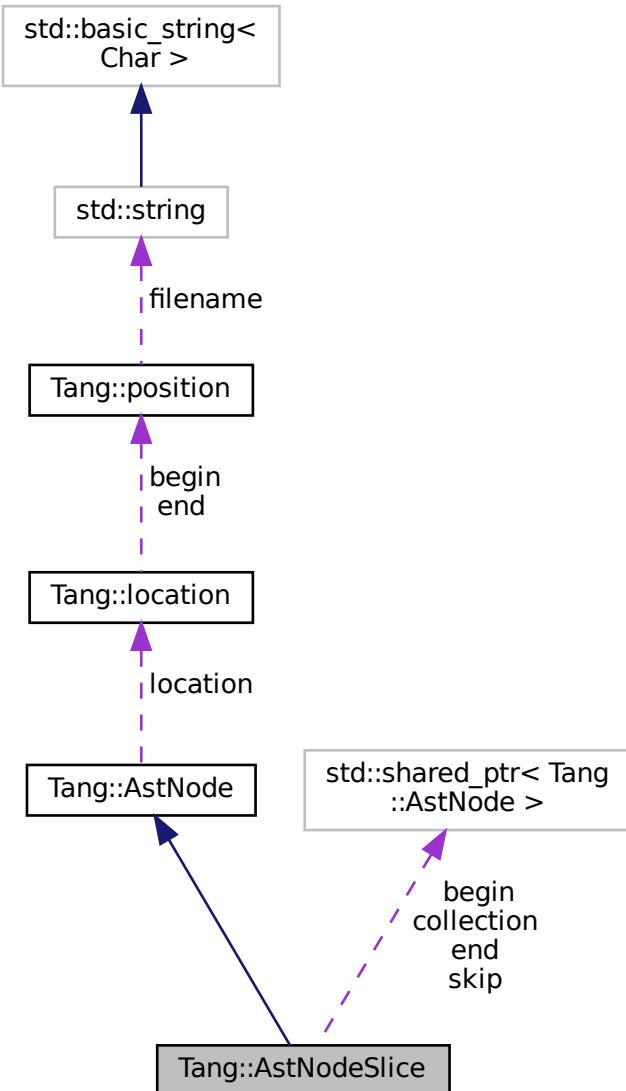
An [AstNode](#) that represents a ternary expression.

```
#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 (shared_ptr< AstNode > collection, shared_ptr< AstNode > begin, shared_ptr< AstNode > end, 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

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

5.24.1 Detailed Description

An `AstNode` that represents a ternary expression.

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

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

5.24.3 Constructor & Destructor Documentation

5.24.3.1 AstNodeSlice()

```
AstNodeSlice::AstNodeSlice (
    shared_ptr< AstNode > collection,
    shared_ptr< AstNode > begin,
    shared_ptr< AstNode > end,
    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.

5.24.4 Member Function Documentation

5.24.4.1 compile()

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

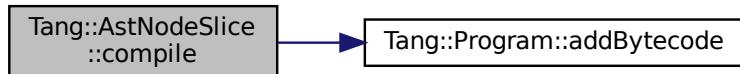
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.24.4.2 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](#).

5.24.4.3 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:

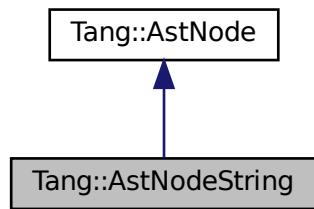
- [include/astNodeSlice.hpp](#)
- [src/astNodeSlice.cpp](#)

5.25 Tang::AstNodeString Class Reference

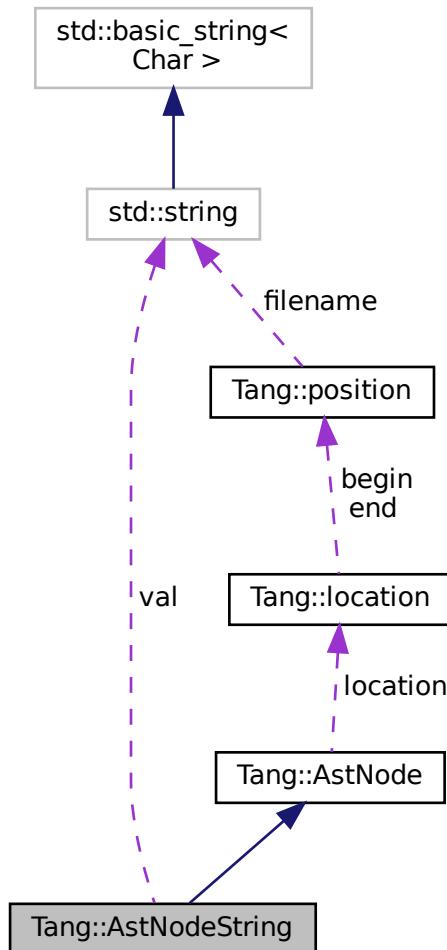
An [AstNode](#) that represents a string literal.

```
#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 string &text, 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.
- `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.

5.25.1 Detailed Description

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

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

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

5.25.3 Constructor & Destructor Documentation

5.25.3.1 AstNodeString()

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

The constructor.

Parameters

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

5.25.4 Member Function Documentation

5.25.4.1 compile()

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

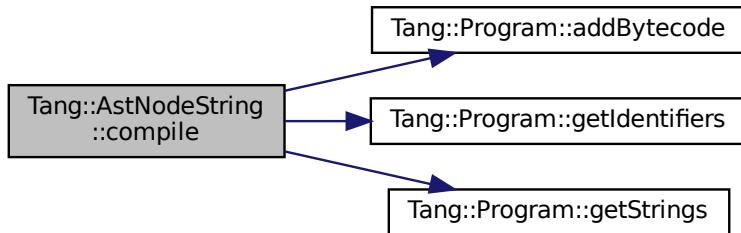
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.25.4.2 compileLiteral()

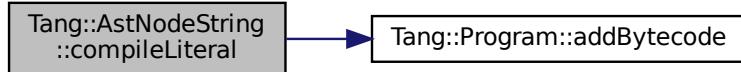
```
void AstNodeString::compileLiteral (
    Tang::Program & program ) const
```

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:



5.25.4.3 compilePreprocess()

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

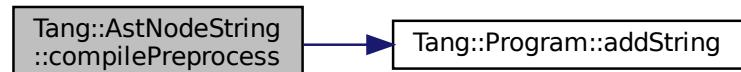
Run any preprocess analysis needed before compilation.

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:



5.25.4.4 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:

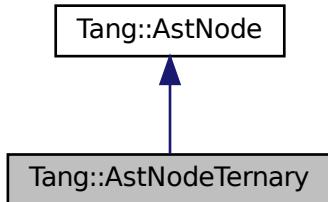
- [include/astNodeString.hpp](#)
- [src/astNodeString.cpp](#)

5.26 Tang::AstNodeTernary Class Reference

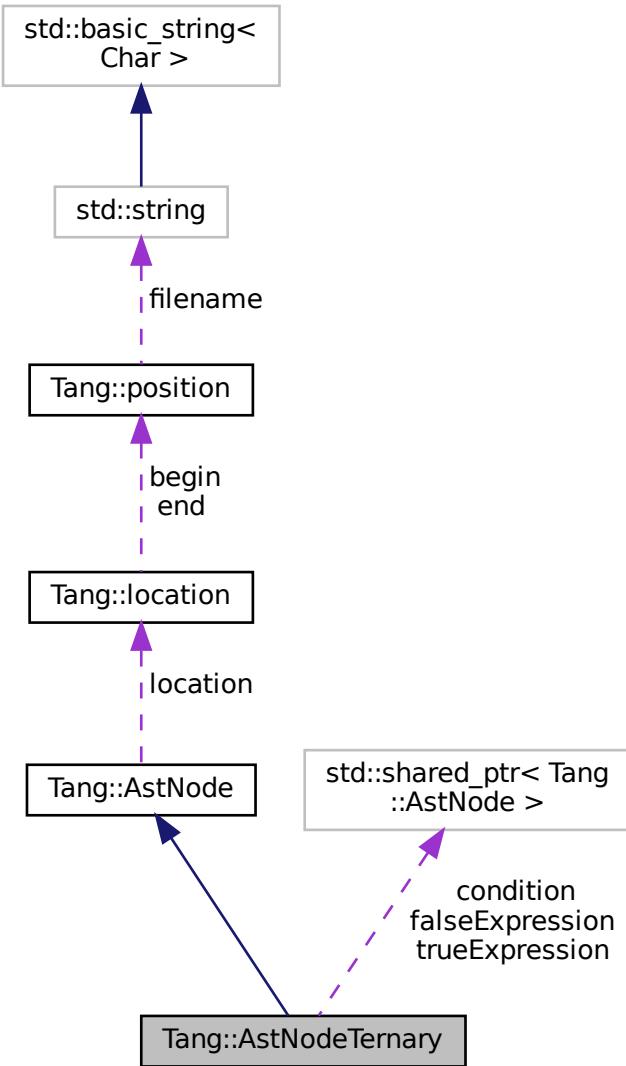
An [AstNode](#) that represents a ternary expression.

```
#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** (`shared_ptr< AstNode > condition, shared_ptr< AstNode > trueExpression, 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

- `shared_ptr< AstNode > condition`
The expression which determines whether the trueExpression or falseExpression is executed.
- `shared_ptr< AstNode > trueExpression`
The expression executed when the condition is true.
- `shared_ptr< AstNode > falseExpression`
The expression executed when the condition is false.

5.26.1 Detailed Description

An `AstNode` that represents a ternary expression.

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 AstNodeTernary()

```
AstNodeTernary::AstNodeTernary (
    shared_ptr< AstNode > condition,
    shared_ptr< AstNode > trueExpression,
    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.

5.26.4 Member Function Documentation

5.26.4.1 compile()

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

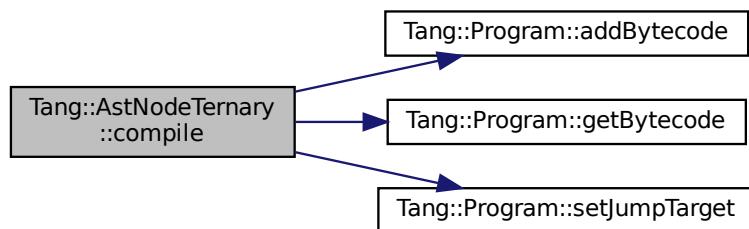
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.26.4.2 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](#).

5.26.4.3 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:

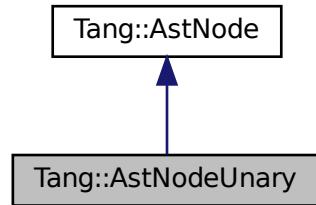
- [include/astNodeTernary.hpp](#)
- [src/astNodeTernary.cpp](#)

5.27 Tang::AstNodeUnary Class Reference

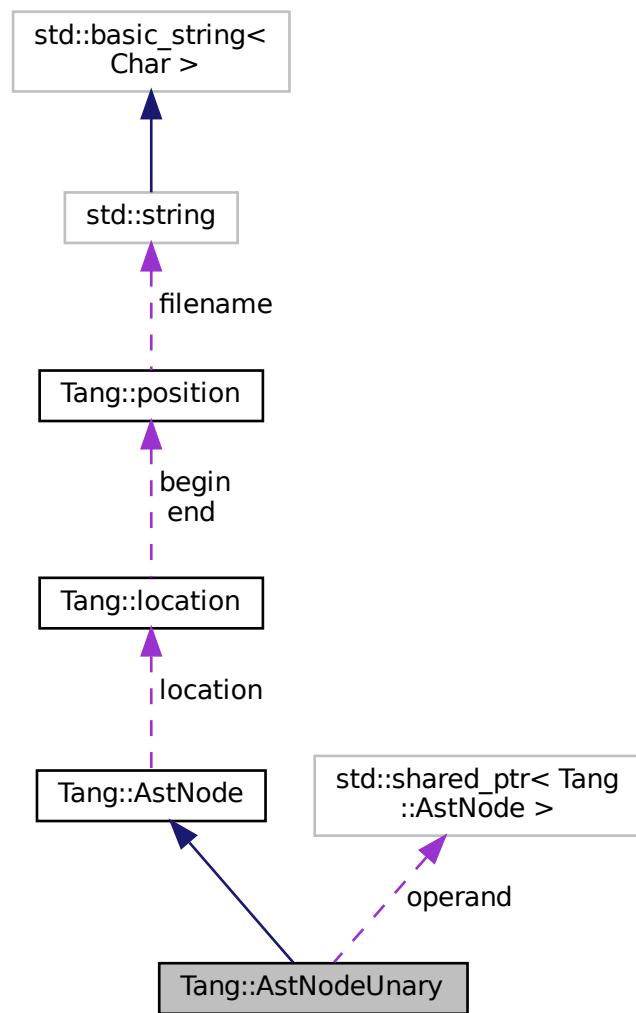
An [AstNode](#) that represents a unary negation.

```
#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, 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.
- `shared_ptr< AstNode > operand`
The operand to which the operation will be applied.

5.27.1 Detailed Description

An `AstNode` that represents a unary negation.

5.27.2 Member Enumeration Documentation

5.27.2.1 Operator

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

The type of operation.

Enumerator

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

5.27.2.2 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.

5.27.3 Constructor & Destructor Documentation**5.27.3.1 AstNodeUnary()**

```
AstNodeUnary::AstNodeUnary (
    Operator op,
    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.

5.27.4 Member Function Documentation**5.27.4.1 compile()**

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

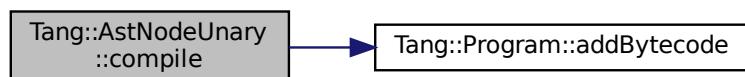
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.27.4.2 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](#).

5.27.4.3 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:

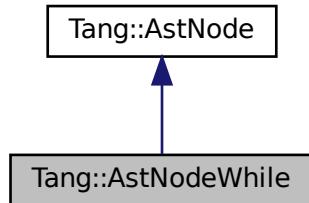
- [include/astNodeUnary.hpp](#)
- [src/astNodeUnary.cpp](#)

5.28 Tang::AstNodeWhile Class Reference

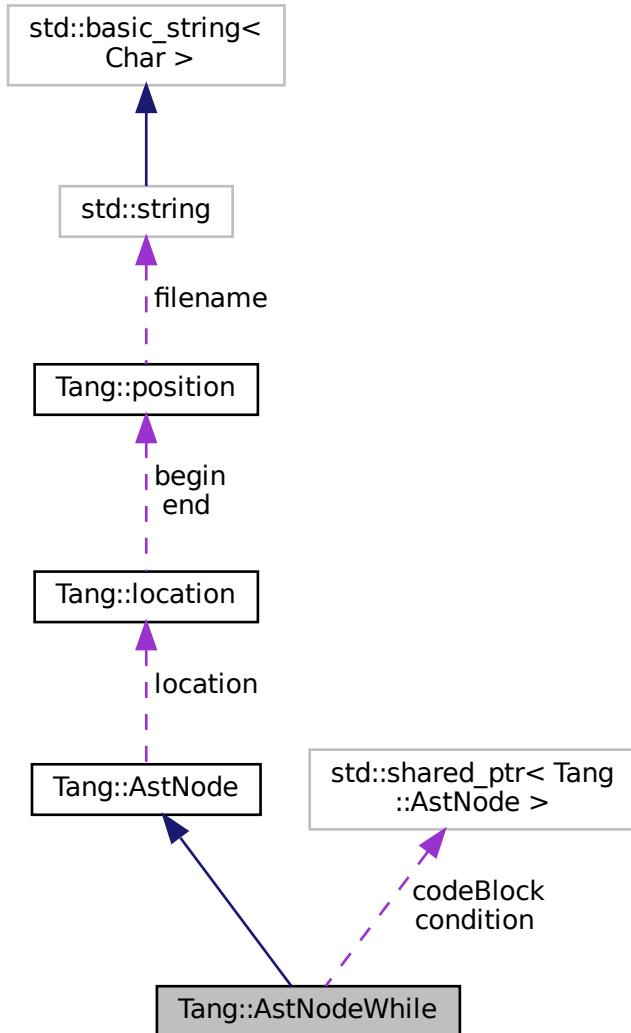
An [AstNode](#) that represents a while statement.

```
#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** (`shared_ptr< AstNode > condition, shared_ptr< AstNode > codeBlock, 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.

Protected Attributes

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

Private Attributes

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

5.28.1 Detailed Description

An `AstNode` that represents a while statement.

5.28.2 Member Enumeration Documentation

5.28.2.1 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.

5.28.3 Constructor & Destructor Documentation

5.28.3.1 AstNodeWhile()

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

```
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.

5.28.4 Member Function Documentation

5.28.4.1 compile()

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

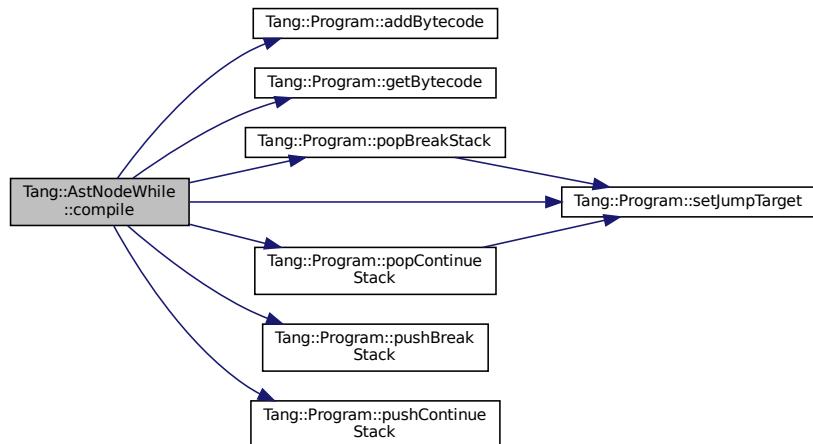
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.28.4.2 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](#).

5.28.4.3 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:

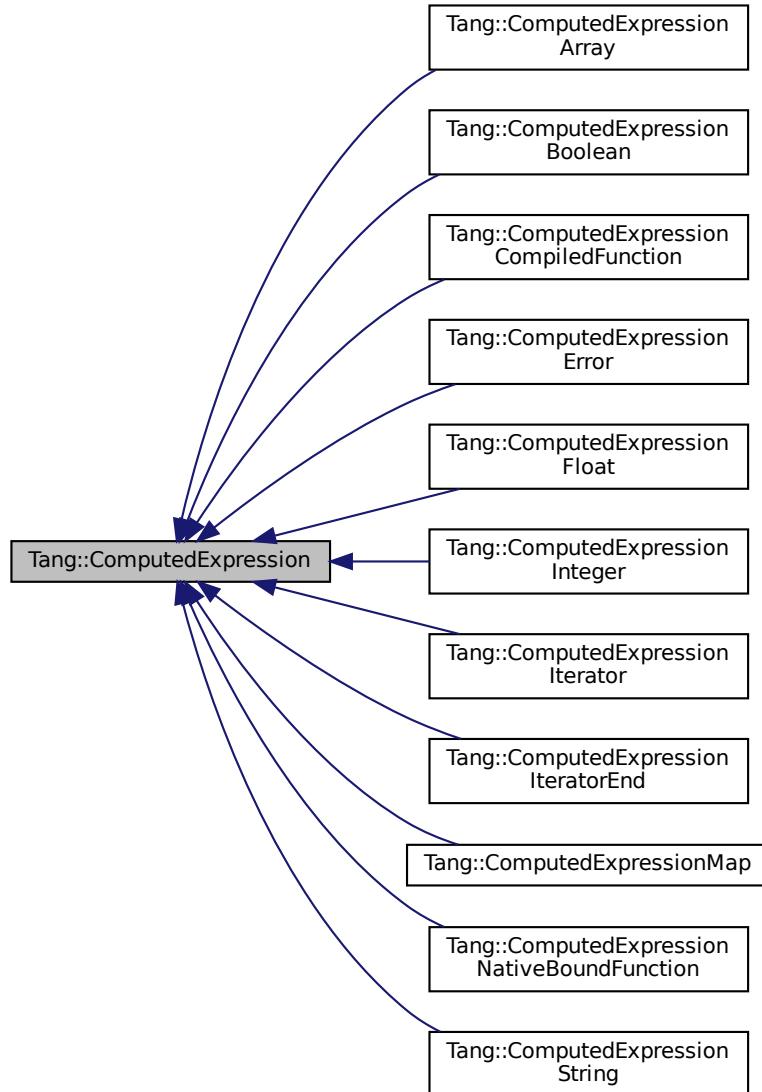
- [include/astNodeWhile.hpp](#)
- [src/astNodeWhile.cpp](#)

5.29 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 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.

Static Public Member Functions

- static [GarbageCollected nativeBoundArgumentCountError \(\)](#)
Provide a standard error message for an unexpected number of arguments provided to a NativeBoundFunction.
- static [GarbageCollected nativeBoundTypeMismatchError \(\)](#)
Provide a standard error message for a type mismatch between a NativeBoundFunction to a [ComputedExpression](#).

5.29.1 Detailed Description

Represents the result of a computation that has been executed.

By default, it will represent a NULL value.

5.29.2 Member Function Documentation

5.29.2.1 [__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](#).

5.29.2.2 [__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](#).

5.29.2.3 `__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](#).

5.29.2.4 `__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](#).

5.29.2.5 `__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](#).

5.29.2.6 `__equal()`

```
GarbageCollected ComputedExpression::__equal (
    const GarbageCollected & rhs ) const [virtual]
```

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::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), and [Tang::ComputedExpressionBoolean](#).

5.29.2.7 `__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](#).

5.29.2.8 `__getIterator()`

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [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 in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.29.2.9 __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](#).

5.29.2.10 __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](#).

5.29.2.11 __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](#).

5.29.2.12 `__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](#).

5.29.2.13 `__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](#).

5.29.2.14 `__multiply()`

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [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 in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.29.2.15 `__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](#).

5.29.2.16 `__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](#).

5.29.2.17 `__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.

5.29.2.18 `__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](#).

5.29.2.19 `__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](#).

5.29.2.20 `__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 <code>GarbageCollected</code> value to subtract from this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.29.2.21 `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::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpression](#), [Tang::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionIterator](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), [Tang::ComputedExpressionBoolean](#), and [Tang::ComputedExpressionArray](#).

5.29.2.22 `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](#).

5.29.2.23 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

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.29.2.24 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

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.29.2.25 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const string & 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](#).

5.29.2.26 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](#).

5.29.2.27 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](#).

5.29.2.28 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](#).

5.29.2.29 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::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpression](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionBoolean](#), and [Tang::ComputedExpressionArray](#).

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

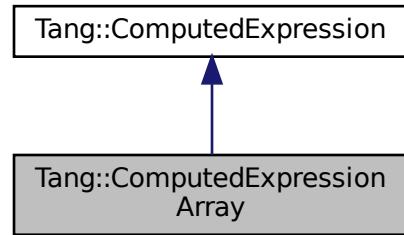
- [include/computedExpression.hpp](#)
- [src/computedExpression.cpp](#)

5.30 Tang::ComputedExpressionArray Class Reference

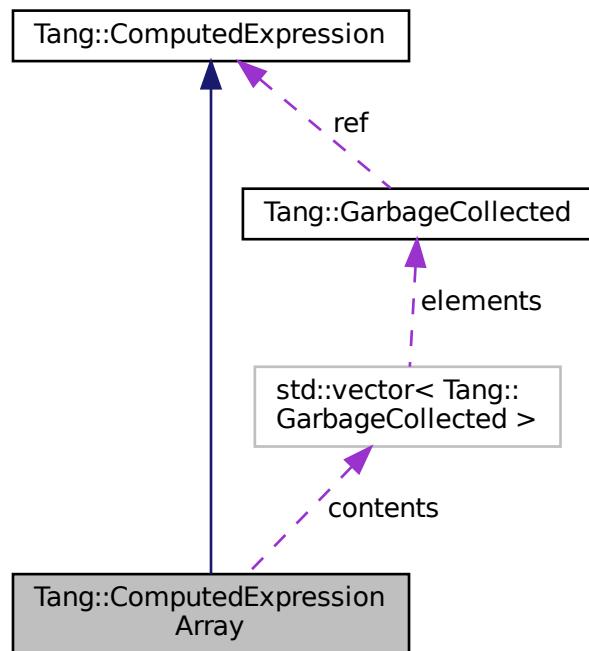
Represents an Array that is the result of a computation.

```
#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) const override`
Perform an index assignment to the supplied value.
- virtual `GarbageCollected __string () const override`
Perform a type cast to string.
- 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 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`

- 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.
- static `GarbageCollected nativeBoundArgumentCountError` ()

Provide a standard error message for an unexpected number of arguments provided to a NativeBoundFunction.
- static `GarbageCollected nativeBoundTypeMismatchError` ()

Provide a standard error message for a type mismatch between a NativeBoundFunction to a `ComputedExpression`.

Private Attributes

- `std::vector< Tang::GarbageCollected > contents`

The array contents.

5.30.1 Detailed Description

Represents an Array that is the result of a computation.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 `ComputedExpressionArray()`

```
ComputedExpressionArray::ComputedExpressionArray (
    std::vector< Tang::GarbageCollected > contents )
```

Construct an Array result.

Parameters

<code>val</code>	The integer value.
------------------	--------------------

5.30.3 Member Function Documentation

5.30.3.1 __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](#).

5.30.3.2 __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](#).

5.30.3.3 __assign_index()

```
GarbageCollected ComputedExpressionArray::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [override], [virtual]
```

Perform an index assignment to the supplied value.

Parameters

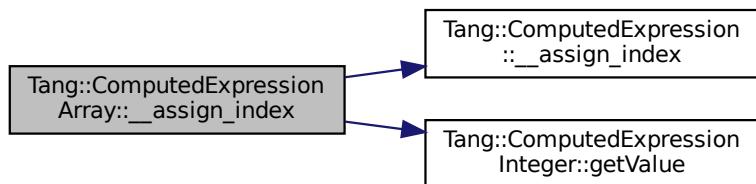
<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

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

Here is the call graph for this function:

**5.30.3.4 __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](#).

5.30.3.5 __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](#).

5.30.3.6 `__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::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), and [Tang::ComputedExpressionBoolean](#).

5.30.3.7 `__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](#).

5.30.3.8 `__getIterator()`

```
GarbageCollected ComputedExpressionArray::__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](#).

5.30.3.9 `__index()`

```
GarbageCollected ComputedExpressionArray::__index (
    const GarbageCollected & index ) const [override], [virtual]
```

Perform an index operation.

Parameters

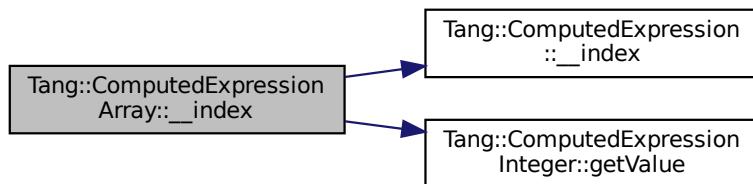
<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:



5.30.3.10 `__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](#).

5.30.3.11 `__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](#).

5.30.3.12 `__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](#).

5.30.3.13 `__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](#).

5.30.3.14 `__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](#).

5.30.3.15 `__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](#).

5.30.3.16 `__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](#).

5.30.3.17 `__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.

5.30.3.18 `__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

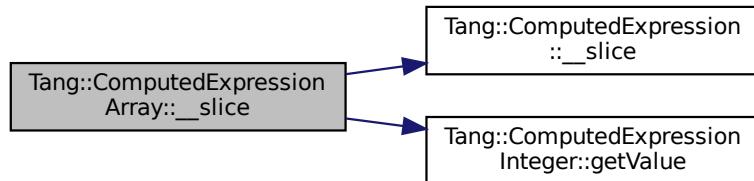
<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

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

Here is the call graph for this function:



5.30.3.19 `__string()`

```
GarbageCollected ComputedExpressionArray::__string ( ) const [override], [virtual]
```

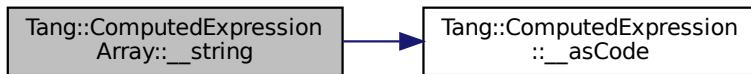
Perform a type cast to string.

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.30.3.20 `__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](#).

5.30.3.21 `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.
-------------	--

5.30.3.22 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](#).

5.30.3.23 getContents()

```
const std::vector< Tang::GarbageCollected > & ComputedExpressionArray::getContents ( ) const
```

Return the contents of this object.

Returns

The contents of this object.

5.30.3.24 getMethods()

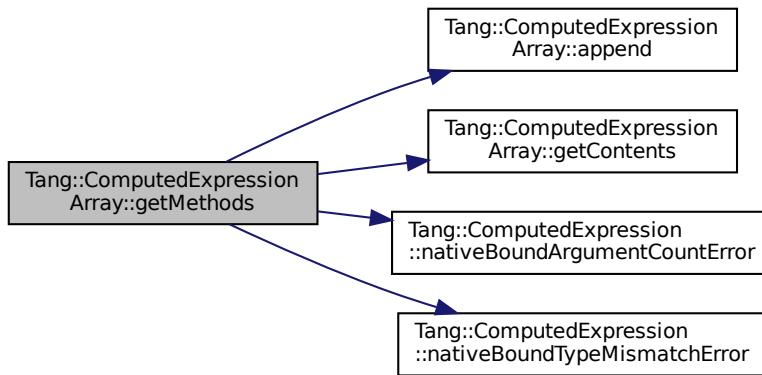
```
NativeBoundFunctionMap ComputedExpressionArray::getMethods ( ) [static]
```

Return the member functions implemented for this particular expression type.

Returns

The member functions implemented.

Here is the call graph for this function:

**5.30.3.25 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](#).

5.30.3.26 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

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.30.3.27 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.

5.30.3.28 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const 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](#).

5.30.3.29 `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

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.30.3.30 `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](#).

5.30.3.31 `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](#).

5.30.3.32 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:

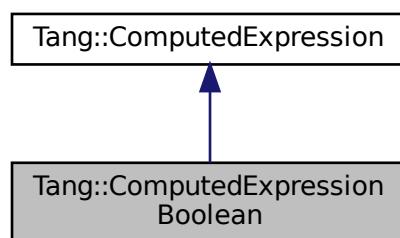
- `include/computedExpressionArray.hpp`
- `src/computedExpressionArray.cpp`

5.31 Tang::ComputedExpressionBoolean Class Reference

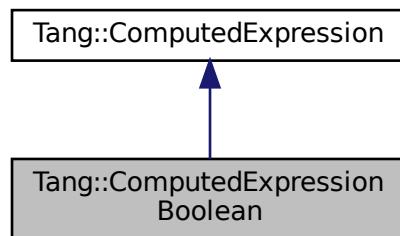
Represents an Boolean that is the result of a computation.

```
#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 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.

Static Public Member Functions

- static `GarbageCollected nativeBoundArgumentCountError` ()
Provide a standard error message for an unexpected number of arguments provided to a NativeBoundFunction.
- static `GarbageCollected nativeBoundTypeMismatchError` ()
Provide a standard error message for a type mismatch between a NativeBoundFunction to a ComputedExpression.

Private Attributes

- bool `val`
The boolean value.

5.31.1 Detailed Description

Represents an Boolean that is the result of a computation.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 ComputedExpressionBoolean()

```
ComputedExpressionBoolean::ComputedExpressionBoolean (
    bool val )
```

Construct an Boolean result.

Parameters

<code>val</code>	The boolean value.
------------------	--------------------

5.31.3 Member Function Documentation

5.31.3.1 __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](#).

5.31.3.2 __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](#).

5.31.3.3 __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](#).

5.31.3.4 `__boolean()`

```
GarbageCollected ComputedExpressionBoolean::__boolean ( ) const [override], [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.31.3.5 `__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](#).

5.31.3.6 `__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:

**5.31.3.7 __float()**

[GarbageCollected](#) `ComputedExpressionBoolean::__float () const [override], [virtual]`

Perform a type cast to float.

Returns

The result of the the operation.

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

5.31.3.8 __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](#).

5.31.3.9 __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](#).

5.31.3.10 __integer()

```
GarbageCollected ComputedExpressionBoolean::__integer ( ) const [override], [virtual]
```

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.31.3.11 __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](#).

5.31.3.12 `__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](#).

5.31.3.13 `__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](#).

5.31.3.14 `__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](#).

5.31.3.15 [__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](#).

5.31.3.16 [__not\(\)](#)

[GarbageCollected](#) `ComputedExpressionBoolean::__not () const [override], [virtual]`

Compute the logical not of this value.

Returns

The result of the operation.

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

5.31.3.17 [__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.

5.31.3.18 `__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](#).

5.31.3.19 `__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](#).

5.31.3.20 `__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](#).

5.31.3.21 `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](#).

5.31.3.22 `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](#).

5.31.3.23 `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](#).

5.31.3.24 `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.

5.31.3.25 `is_equal()` [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const 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](#).

5.31.3.26 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](#).

5.31.3.27 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](#).

5.31.3.28 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`.

5.31.3.29 `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:

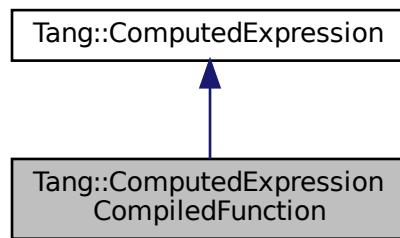
- `include/computedExpressionBoolean.hpp`
- `src/computedExpressionBoolean.cpp`

5.32 `Tang::ComputedExpressionCompiledFunction` Class Reference

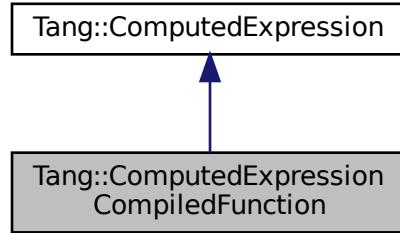
Represents a Compiled Function declared in the script.

```
#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 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.

Static Public Member Functions

- static `GarbageCollected nativeBoundArgumentCountError` ()
Provide a standard error message for an unexpected number of arguments provided to a NativeBoundFunction.
- static `GarbageCollected nativeBoundTypeMismatchError` ()
Provide a standard error message for a type mismatch between a NativeBoundFunction to a ComputedExpression.

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.

5.32.1 Detailed Description

Represents a Compiled Function declared in the script.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 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.

5.32.3 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

<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](#).

5.32.3.2 `__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 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](#).

5.32.3.4 `__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 ComputedExpressionCompiledFunction::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

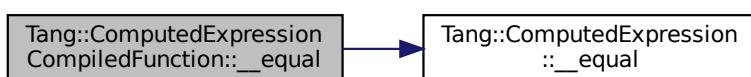
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.32.3.7 [__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 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](#).

5.32.3.9 `__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](#).

5.32.3.10 `__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 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](#).

5.32.3.12 __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](#).

5.32.3.13 __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 GarbageCollected 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.

5.32.3.18 `__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](#).

5.32.3.19 `__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](#).

5.32.3.20 `__subtract()`

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<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](#).

5.32.3.21 `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](#).

5.32.3.22 `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](#).

5.32.3.23 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

val	The value to compare against.
-----	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.32.3.24 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.

5.32.3.25 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const 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](#).

5.32.3.26 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](#).

5.32.3.27 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](#).

5.32.3.28 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](#).

5.32.3.29 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:

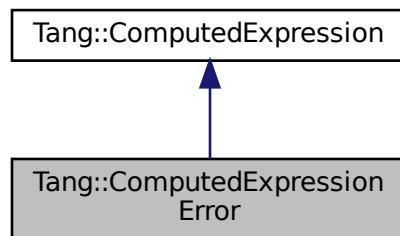
- [include/computedExpressionCompiledFunction.hpp](#)
- [src/computedExpressionCompiledFunction.cpp](#)

5.33 Tang::ComputedExpressionError Class Reference

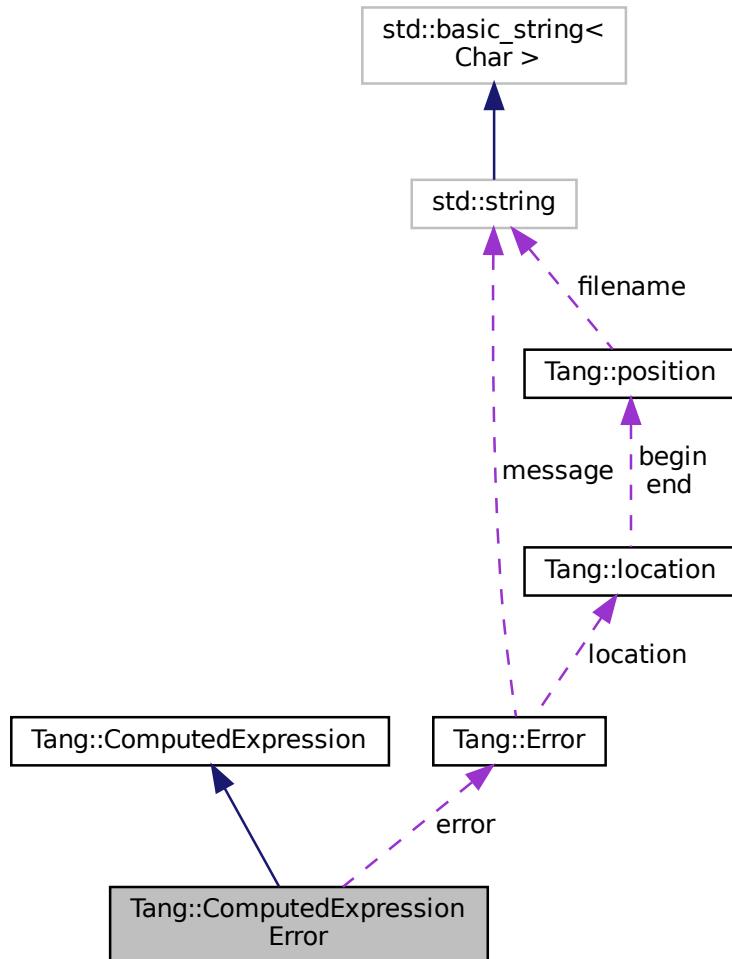
Represents a Runtime [Error](#).

```
#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 multiplying this value and the supplied value.
- virtual `GarbageCollected __modulo` (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 moduloing this value and the supplied value.
- virtual `GarbageCollected __not` () const override

Compute the result of negating 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 `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.

Static Public Member Functions

- static [GarbageCollected nativeBoundArgumentCountError \(\)](#)
Provide a standard error message for an unexpected number of arguments provided to a NativeBoundFunction.
- static [GarbageCollected nativeBoundTypeMismatchError \(\)](#)
Provide a standard error message for a type mismatch between a NativeBoundFunction to a [ComputedExpression](#).

Private Attributes

- [Tang::Error error](#)
The [Error](#) object.

5.33.1 Detailed Description

Represents a Runtime [Error](#).

5.33.2 Constructor & Destructor Documentation

5.33.2.1 [ComputedExpressionError\(\)](#)

```
ComputedExpressionError::ComputedExpressionError (
    Tang::Error error )
```

Construct a Runtime [Error](#).

Parameters

error	The Tang::Error object.
-----------------------	---

5.33.3 Member Function Documentation

5.33.3.1 [__add\(\)](#)

```
GarbageCollected ComputedExpressionError::__add (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of adding this value and the supplied value.

Parameters

rhs	The GarbageCollected value to add to this.
---------------------	--

Returns

The result of the operation.

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

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 ComputedExpressionError::__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 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](#).

5.33.3.6 `__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](#).

5.33.3.7 `__float()`

```
GarbageCollected ComputedExpressionError::__float ( ) const [override], [virtual]
```

Perform a type cast to float.

Returns

The result of the the operation.

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

5.33.3.8 `__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](#).

5.33.3.9 `__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 ComputedExpressionError::__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 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](#).

5.33.3.13 `__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](#).

5.33.3.14 `__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 GarbageCollected value to multiply to this.
------------------	---

Returns

The result of the operation.

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

5.33.3.15 `__negative()`

```
GarbageCollected ComputedExpressionError::__negative () const [override], [virtual]
```

Compute the result of negating this value.

Returns

The result of the operation.

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

5.33.3.16 `__not()`

```
GarbageCollected ComputedExpressionError::__not () const [override], [virtual]
```

Compute the logical not of this value.

Returns

The result of the operation.

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

5.33.3.17 `__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.

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 ComputedExpressionError::__string ( ) const [override], [virtual]
```

Perform a type cast to string.

Returns

The result of the the operation.

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

5.33.3.20 `__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 GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

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

5.33.3.21 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](#).

5.33.3.22 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](#).

5.33.3.23 `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

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

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

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

<i>val</i>	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 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](#).

5.33.3.26 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

val	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

val	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` `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:

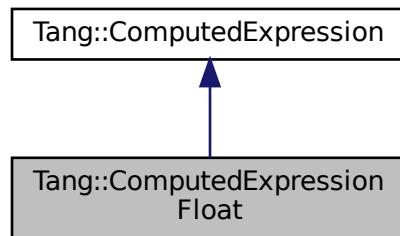
- `include/computedExpressionError.hpp`
- `src/computedExpressionError.cpp`

5.34 `Tang::ComputedExpressionFloat` Class Reference

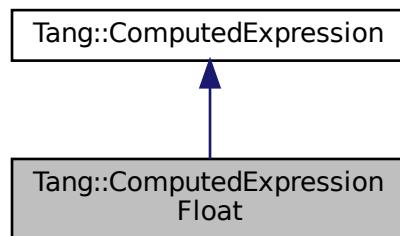
Represents a `Float` that is the result of a computation.

```
#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 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)**

- virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const

Perform an index assignment to the supplied value.

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.

Static Public Member Functions

- static `GarbageCollected nativeBoundArgumentCountError` ()

Provide a standard error message for an unexpected number of arguments provided to a NativeBoundFunction.
- static `GarbageCollected nativeBoundTypeMismatchError` ()

Provide a standard error message for a type mismatch between a NativeBoundFunction to a `ComputedExpression`.

Private Attributes

- `Tang::float_t val`

The float value.

5.34.1 Detailed Description

Represents a Float that is the result of a computation.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 `ComputedExpressionFloat()`

```
ComputedExpressionFloat::ComputedExpressionFloat (
    Tang::float_t val )
```

Construct a Float result.

Parameters

<code>val</code>	The float value.
------------------	------------------

5.34.3 Member Function Documentation

5.34.3.1 __add()

```
GarbageCollected ComputedExpressionFloat::__add (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of adding this value and the supplied value.

Parameters

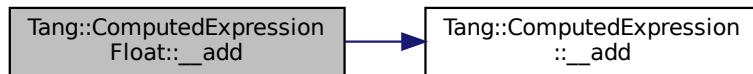
<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:



5.34.3.2 __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` `ComputedExpressionFloat::__boolean () const [override], [virtual]`

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.34.3.5 `__divide()`

`GarbageCollected` `ComputedExpressionFloat::__divide (`
`const GarbageCollected & rhs) const [override], [virtual]`

Compute the result of dividing this value and the supplied value.

Parameters

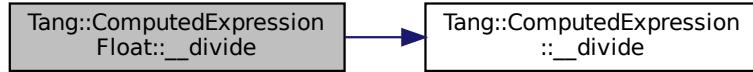
<i>rhs</i>	The <code>GarbageCollected</code> value to divide this by.
------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:



5.34.3.6 __equal()

```
GarbageCollected ComputedExpressionFloat::__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 ComputedExpressionFloat::__float ( ) const [override], [virtual]
```

Perform a type cast to float.

Returns

The result of the the operation.

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

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 <code>GarbageCollected</code> 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 ComputedExpressionFloat::__integer ( ) const [override], [virtual]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented from `Tang::ComputedExpression`.

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 ComputedExpressionFloat::__lessThan (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the "less than" comparison.

Parameters

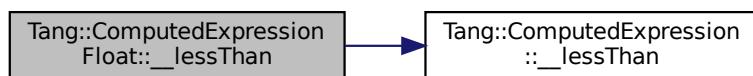
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.34.3.13 `__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 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:

**5.34.3.15 __negative()**

```
GarbageCollected ComputedExpressionFloat::__negative ( ) const [override], [virtual]
```

Compute the result of negating this value.

Returns

The result of the operation.

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

5.34.3.16 __not()

```
GarbageCollected ComputedExpressionFloat::__not ( ) const [override], [virtual]
```

Compute the logical not of this value.

Returns

The result of the operation.

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

5.34.3.17 __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.

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 ComputedExpressionFloat::__string ( ) const [override], [virtual]
```

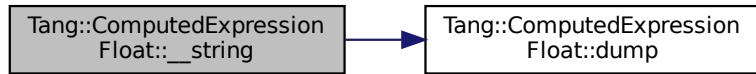
Perform a type cast to string.

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.34.3.20 `__subtract()`

```
GarbageCollected ComputedExpressionFloat::__subtract (   
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of subtracting this value and the supplied value.

Parameters

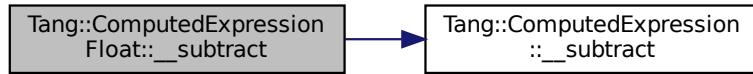
<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.34.3.21 `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](#).

5.34.3.22 `getValue()`

```
Tang::float_t ComputedExpressionFloat::getValue ( ) const
```

Helper function to get the value associated with this expression.

Returns

The value associated with this expression.

5.34.3.23 `is_equal()` [1/6]

```
bool ComputedExpressionFloat::is_equal ( \n    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](#).

5.34.3.24 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](#).

5.34.3.25 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.

5.34.3.26 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const 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](#).

5.34.3.27 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](#).

5.34.3.28 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](#).

5.34.3.29 `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](#).

5.34.3.30 `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:

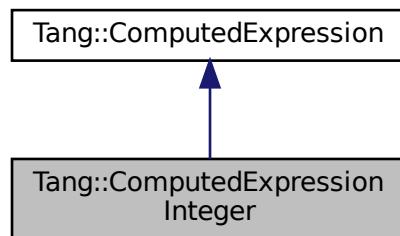
- [include/computedExpressionFloat.hpp](#)
- [src/computedExpressionFloat.cpp](#)

5.35 [Tang::ComputedExpressionInteger](#) Class Reference

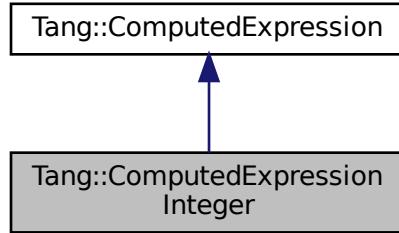
Represents an Integer that is the result of a computation.

```
#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 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.

Static Public Member Functions

- static `GarbageCollected nativeBoundArgumentCountError ()`
Provide a standard error message for an unexpected number of arguments provided to a NativeBoundFunction.
- static `GarbageCollected nativeBoundTypeMismatchError ()`
Provide a standard error message for a type mismatch between a NativeBoundFunction to a `ComputedExpression`.

Private Attributes

- `Tang::integer_t val`
The integer value.

5.35.1 Detailed Description

Represents an Integer that is the result of a computation.

5.35.2 Constructor & Destructor Documentation

5.35.2.1 ComputedExpressionInteger()

```
ComputedExpressionInteger::ComputedExpressionInteger ( Tang::integer_t val )
```

Construct an Integer result.

Parameters

<code>val</code>	The integer value.
------------------	--------------------

5.35.3 Member Function Documentation

5.35.3.1 __add()

```
GarbageCollected ComputedExpressionInteger::__add ( const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of adding this value and the supplied value.

Parameters

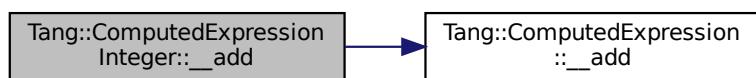
<code>rhs</code>	The <code>GarbageCollected</code> value to add to this.
------------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



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 ComputedExpressionInteger::__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 ComputedExpressionInteger::__divide (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of dividing this value and the supplied value.

Parameters

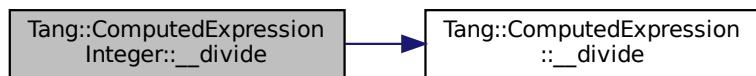
<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.35.3.6 __equal()

```

GarbageCollected ComputedExpressionInteger::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]

```

Perform an equality test.

Parameters

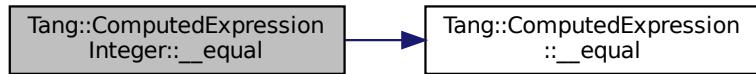
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.35.3.7 `__float()`

```
GarbageCollected ComputedExpressionInteger::__float ( ) const [override], [virtual]
```

Perform a type cast to float.

Returns

The result of the the operation.

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

5.35.3.8 `__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 ComputedExpressionInteger::__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 ComputedExpressionInteger::__lessThan ( const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the "less than" comparison.

Parameters

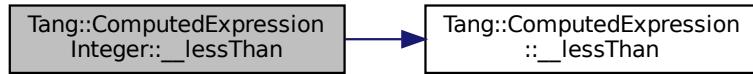
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.35.3.13 __modulo()

```
GarbageCollected ComputedExpressionInteger::__modulo (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of moduloing this value and the supplied value.

Parameters

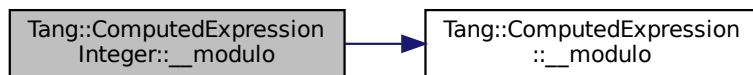
<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.35.3.14 __multiply()

```
GarbageCollected ComputedExpressionInteger::__multiply (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of multiplying this value and the supplied value.

Parameters

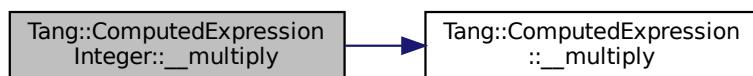
<code>rhs</code>	The GarbageCollected value to multiply to this.
------------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:

**5.35.3.15 __negative()**

[GarbageCollected](#) `ComputedExpressionInteger::__negative () const [override], [virtual]`

Compute the result of negating this value.

Returns

The result of the operation.

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

5.35.3.16 __not()

[GarbageCollected](#) `ComputedExpressionInteger::__not () const [override], [virtual]`

Compute the logical not of this value.

Returns

The result of the operation.

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

5.35.3.17 __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.

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 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:



5.35.3.20 `__subtract()`

```
GarbageCollected ComputedExpressionInteger::__subtract (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of subtracting this value and the supplied value.

Parameters

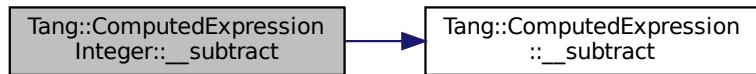
<code>rhs</code>	The GarbageCollected value to subtract from this.
------------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.35.3.21 `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](#).

5.35.3.22 `getValue()`

```
Tang::integer_t ComputedExpressionInteger::getValue ( ) const
```

Helper function to get the value associated with this expression.

Returns

The value associated with this expression.

5.35.3.23 `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

<code>val</code>	The value to compare against.
------------------	-------------------------------

Returns

True if equal, false if not.

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

5.35.3.24 `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](#).

5.35.3.25 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.

5.35.3.26 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const 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](#).

5.35.3.27 `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](#).

5.35.3.28 `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](#).

5.35.3.29 `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](#).

5.35.3.30 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:

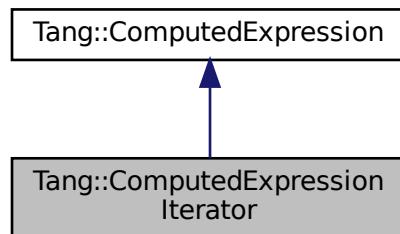
- [include/computedExpressionInteger.hpp](#)
- [src/computedExpressionInteger.cpp](#)

5.36 Tang::ComputedExpressionIterator Class Reference

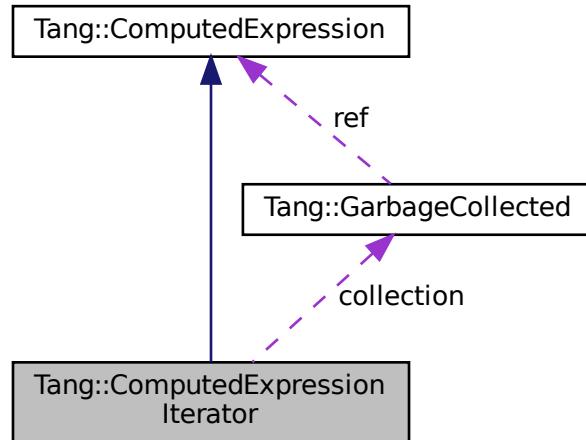
Represents an iterator that is the result of a computation.

```
#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 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
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 `GarbageCollected nativeBoundArgumentCountError` ()
Provide a standard error message for an unexpected number of arguments provided to a NativeBoundFunction.
- static `GarbageCollected nativeBoundTypeMismatchError` ()
Provide a standard error message for a type mismatch between a NativeBoundFunction to a `ComputedExpression`.

Private Attributes

- `Tang::GarbageCollected collection`
The target collection.
- `size_t index`
The next index.

5.36.1 Detailed Description

Represents an iterator that is the result of a computation.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 ComputedExpressionIterator()

```
ComputedExpressionIterator::ComputedExpressionIterator ( Tang::GarbageCollected collection )
```

Construct an iterator result.

Parameters

<i>collection</i>	The collection through which the iterator processes
-------------------	---

5.36.3 Member Function Documentation

5.36.3.1 __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](#).

5.36.3.2 `__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 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](#).

5.36.3.5 `__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](#).

5.36.3.6 [__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::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionNativeFunction](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), and [Tang::ComputedExpressionBoolean](#).

5.36.3.7 [__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](#).

5.36.3.8 [__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](#).

5.36.3.9 __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](#).

5.36.3.10 __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](#).

5.36.3.11 __iteratorNext()

```
GarbageCollected ComputedExpressionIterator::__iteratorNext (
    size_t index ) const [override], [virtual]
```

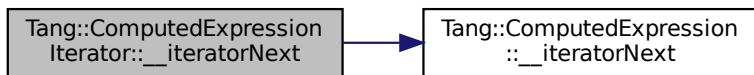
Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

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

Here is the call graph for this function:



5.36.3.12 __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](#).

5.36.3.13 __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 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](#).

5.36.3.15 `__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](#).

5.36.3.16 `__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](#).

5.36.3.17 `__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.

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 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](#).

5.36.3.20 `__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](#).

5.36.3.21 `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](#).

5.36.3.22 `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](#).

5.36.3.23 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](#).

5.36.3.24 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.

5.36.3.25 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const 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](#).

5.36.3.26 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](#).

5.36.3.27 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](#).

5.36.3.28 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](#).

5.36.3.29 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::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionObject](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionObject](#), [Tang::ComputedExpressionBoolean](#), and [Tang::ComputedExpressionArray](#).

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

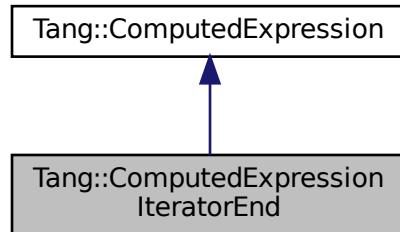
- include/computedExpressionIterator.hpp
- src/computedExpressionIterator.cpp

5.37 Tang::ComputedExpressionIteratorEnd Class Reference

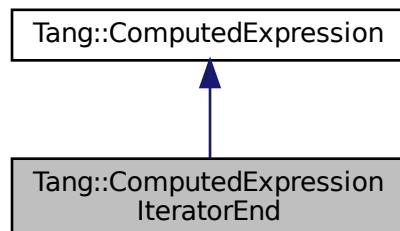
Represents that a collection has no more values through which to iterate.

```
#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 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.

Static Public Member Functions

- static [GarbageCollected nativeBoundArgumentCountError \(\)](#)
Provide a standard error message for an unexpected number of arguments provided to a NativeBoundFunction.
- static [GarbageCollected nativeBoundTypeMismatchError \(\)](#)
Provide a standard error message for a type mismatch between a NativeBoundFunction to a [ComputedExpression](#).

5.37.1 Detailed Description

Represents that a collection has no more values through which to iterate.

5.37.2 Member Function Documentation

5.37.2.1 [__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 GarbageCollected value to add to this.
------------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.37.2.2 [__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](#).

5.37.2.3 `__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](#).

5.37.2.4 `__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](#).

5.37.2.5 `__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](#).

5.37.2.6 `__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::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), and [Tang::ComputedExpressionBoolean](#).

5.37.2.7 `__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](#).

5.37.2.8 `__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](#).

5.37.2.9 __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](#).

5.37.2.10 __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](#).

5.37.2.11 __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](#).

5.37.2.12 `__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](#).

5.37.2.13 `__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](#).

5.37.2.14 `__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 <code>GarbageCollected</code> value to multiply to this.
------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.37.2.15 `__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](#).

5.37.2.16 `__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](#).

5.37.2.17 `__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.

5.37.2.18 `__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](#).

5.37.2.19 `__string()`

```
GarbageCollected ComputedExpressionIteratorEnd::__string ( ) const [override], [virtual]
```

Perform a type cast to string.

Returns

The result of the the operation.

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

5.37.2.20 `__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](#).

5.37.2.21 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](#).

5.37.2.22 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](#).

5.37.2.23 `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](#).

5.37.2.24 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.

5.37.2.25 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const 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](#).

5.37.2.26 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](#).

5.37.2.27 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](#).

5.37.2.28 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](#).

5.37.2.29 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::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpression](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionBoolean](#), and [Tang::ComputedExpressionArray](#).

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

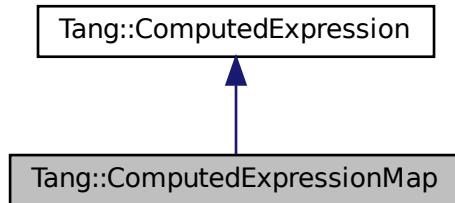
- [include/computedExpressionIteratorEnd.hpp](#)
- [src/computedExpressionIteratorEnd.cpp](#)

5.38 Tang::ComputedExpressionMap Class Reference

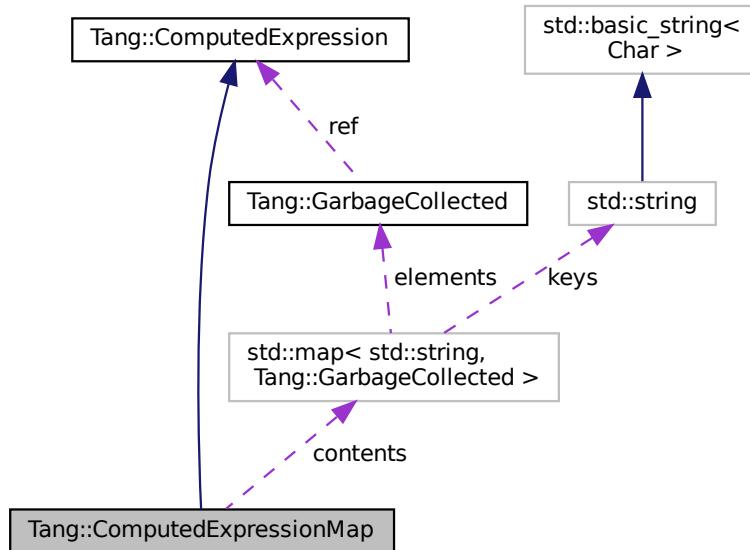
Represents an Map that is the result of a computation.

```
#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**
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 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 `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.

Static Public Member Functions

- **static `GarbageCollected nativeBoundArgumentCountError` ()**

Provide a standard error message for an unexpected number of arguments provided to a NativeBoundFunction.
- **static `GarbageCollected nativeBoundTypeMismatchError` ()**

Provide a standard error message for a type mismatch between a NativeBoundFunction to a `ComputedExpression`.

Private Attributes

- **std::map< std::string, `Tang::GarbageCollected` > contents**

The map contents.

5.38.1 Detailed Description

Represents an Map that is the result of a computation.

5.38.2 Constructor & Destructor Documentation

5.38.2.1 ComputedExpressionMap()

```
ComputedExpressionMap::ComputedExpressionMap ( std::map< std::string, Tang::GarbageCollected > contents )
```

Construct an Map result.

Parameters

<i>contents</i>	The map of key value pairs.
-----------------	-----------------------------

5.38.3 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 ComputedExpressionMap::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [override], [virtual]
```

Perform an index assignment to the supplied value.

Parameters

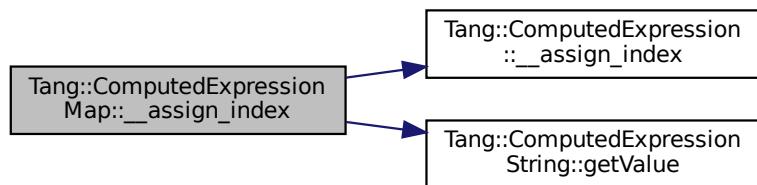
<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

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

Here is the call graph for this function:



5.38.3.4 `__boolean()`

```
GarbageCollected ComputedExpressionMap::__boolean ( ) const [override], [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.38.3.5 `__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::ComputedExpressionNativeBoundFunction](#), [Tang::ComputedExpressionNativeFunction](#), [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 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](#).

5.38.3.9 `__index()`

```
GarbageCollected ComputedExpressionMap::__index ( const GarbageCollected & index ) const [override], [virtual]
```

Perform an index operation.

Parameters

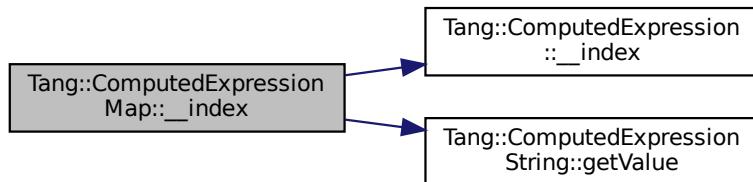
<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:

**5.38.3.10 __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](#) `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](#).

5.38.3.12 `__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.

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 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:



5.38.3.20 `__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 <code>GarbageCollected</code> 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 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](#).

5.38.3.22 `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

val	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

val	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 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](#).

5.38.3.26 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

<i>val</i>	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

<i>val</i>	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 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](#).

5.38.3.29 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:

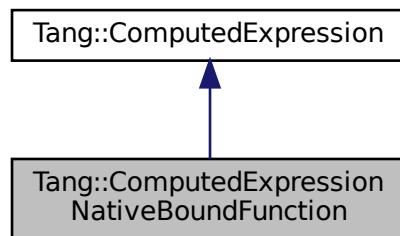
- [include/computedExpressionMap.hpp](#)
- [src/computedExpressionMap.cpp](#)

5.39 Tang::ComputedExpressionNativeBoundFunction Class Reference

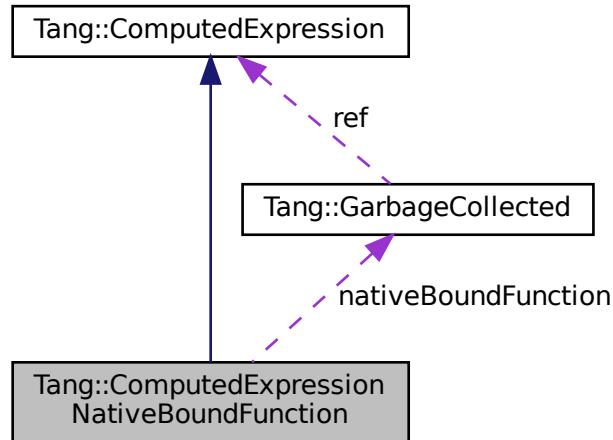
Represents a NativeBound Function declared in the script.

```
#include <computedExpressionNativeBoundFunction.hpp>
```

Inheritance diagram for Tang::ComputedExpressionNativeBoundFunction:



Collaboration diagram for Tang::ComputedExpressionNativeBoundFunction:



Public Member Functions

- **ComputedExpressionNativeBoundFunction** (NativeBoundFunction nativeBoundFunction)
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.
- 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 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.

Static Public Member Functions

- static `GarbageCollected nativeBoundArgumentCountError` ()
Provide a standard error message for an unexpected number of arguments provided to a NativeBoundFunction.
- static `GarbageCollected nativeBoundTypeMismatchError` ()
Provide a standard error message for a type mismatch between a NativeBoundFunction to a `ComputedExpression`.

Public Attributes

- `std::optional< GarbageCollected > target`
The target object that the function is bound to.
- `NativeBoundFunction nativeBoundFunction`
The native bound function to be executed.

5.39.1 Detailed Description

Represents a NativeBound Function declared in the script.

5.39.2 Constructor & Destructor Documentation

5.39.2.1 ComputedExpressionNativeBoundFunction()

```
ComputedExpressionNativeBoundFunction::ComputedExpressionNativeBoundFunction (
```

<code>NativeBoundFunction</code>	<code>nativeBoundFunction</code>
----------------------------------	----------------------------------

```
)
```

Construct an NativeBoundFunction.

Parameters

<code>argc</code>	The count of arguments that this function expects.
<code>pc</code>	The bytecode address of the start of the function.

5.39.3 Member Function Documentation

5.39.3.1 __add()

```
GarbageCollected ComputedExpression::__add (
```

<code>const GarbageCollected</code>	<code>& rhs</code>
-------------------------------------	------------------------

```
) 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](#).

5.39.3.2 `__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](#).

5.39.3.3 `__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](#).

5.39.3.4 `__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](#).

5.39.3.5 `__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](#).

5.39.3.6 [__equal\(\)](#)

```
GarbageCollected ComputedExpressionNativeBoundFunction::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

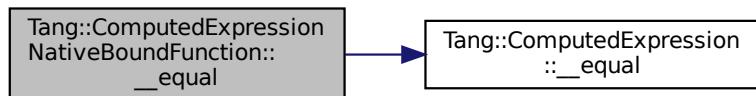
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.39.3.7 [__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](#).

5.39.3.8 `__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](#).

5.39.3.9 `__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](#).

5.39.3.10 `__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](#).

5.39.3.11 `__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](#).

5.39.3.12 `__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](#).

5.39.3.13 `__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](#).

5.39.3.14 [__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](#).

5.39.3.15 [__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](#).

5.39.3.16 `__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](#).

5.39.3.17 `__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.

5.39.3.18 `__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](#).

5.39.3.19 `__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](#).

5.39.3.20 `__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](#).

5.39.3.21 `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](#).

5.39.3.22 `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](#).

5.39.3.23 `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](#).

5.39.3.24 `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.

5.39.3.25 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const 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](#).

5.39.3.26 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](#).

5.39.3.27 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](#).

5.39.3.28 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](#).

5.39.3.29 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:

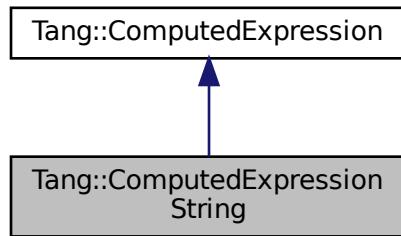
- [include/computedExpressionNativeBoundFunction.hpp](#)
- [src/computedExpressionNativeBoundFunction.cpp](#)

5.40 Tang::ComputedExpressionString Class Reference

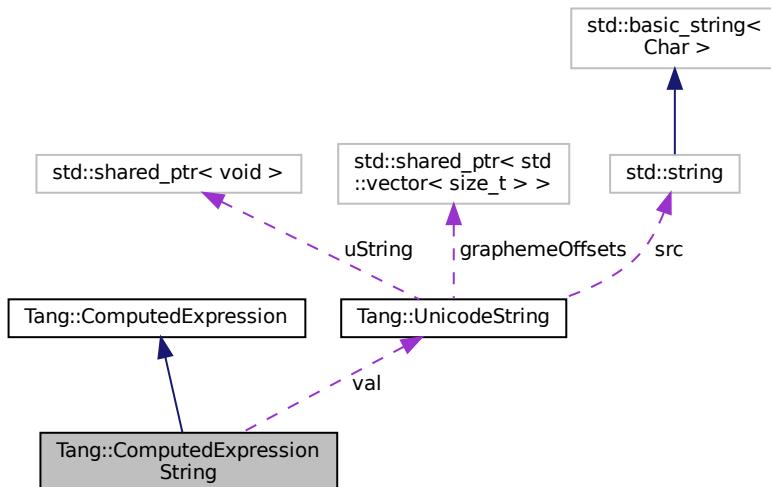
Represents a String that is the result of a computation.

```
#include <computedExpressionString.hpp>
```

Inheritance diagram for Tang::ComputedExpressionString:



Collaboration diagram for Tang::ComputedExpressionString:



Public Member Functions

- [ComputedExpressionString \(std::string val\)](#)
Construct a String result.
- 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 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.
- `UnicodeString getValue () const`
Return the string value that is stored in this object.
- 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) const`
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.
- static `GarbageCollected nativeBoundArgumentCountError ()`
Provide a standard error message for an unexpected number of arguments provided to a NativeBoundFunction.
- static `GarbageCollected nativeBoundTypeMismatchError ()`
Provide a standard error message for a type mismatch between a NativeBoundFunction to a `ComputedExpression`.

Private Attributes

- `UnicodeString val`
The string value.

5.40.1 Detailed Description

Represents a String that is the result of a computation.

5.40.2 Constructor & Destructor Documentation

5.40.2.1 ComputedExpressionString()

```
ComputedExpressionString::ComputedExpressionString ( std::string val )
```

Construct a String result.

Parameters

<code>val</code>	The string value.
------------------	-------------------

5.40.3 Member Function Documentation

5.40.3.1 __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:



5.40.3.2 __asCode()

```
string ComputedExpressionString::__asCode ( ) const [override], [virtual]
```

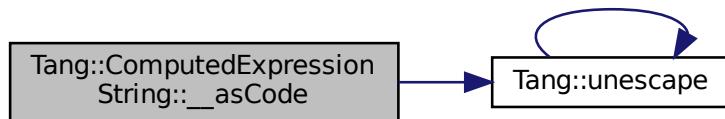
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

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

Here is the call graph for this function:



5.40.3.3 __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](#).

5.40.3.4 __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:



5.40.3.5 `__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](#).

5.40.3.6 `__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:



5.40.3.7 __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](#).

5.40.3.8 __getIterator()

`GarbageCollected` `ComputedExpressionString::__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](#).

5.40.3.9 __index()

```
GarbageCollected ComputedExpressionString::__index (
    const GarbageCollected & index ) const [override], [virtual]
```

Perform an index operation.

Parameters

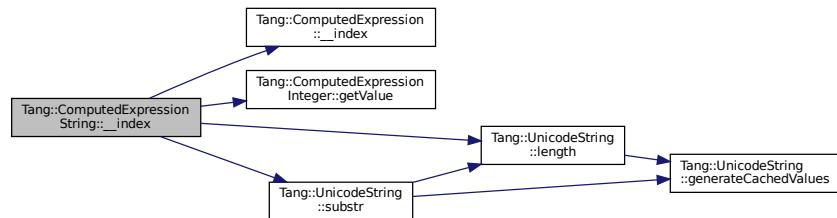
<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:



5.40.3.10 __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](#).

5.40.3.11 __iteratorNext()

```
GarbageCollected ComputedExpressionString::__iteratorNext (
    size_t index ) const [override], [virtual]
```

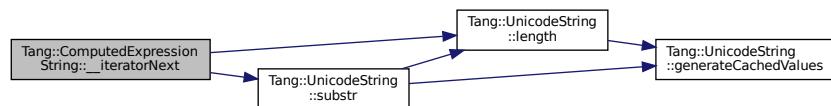
Get the next iterative value.

Parameters

<i>index</i>	The desired index value.
--------------	--------------------------

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

Here is the call graph for this function:

**5.40.3.12 __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:



5.40.3.13 `__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](#).

5.40.3.14 `__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](#).

5.40.3.15 `__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](#).

5.40.3.16 `__not()`

```
GarbageCollected ComputedExpressionString::__not ( ) const [override], [virtual]
```

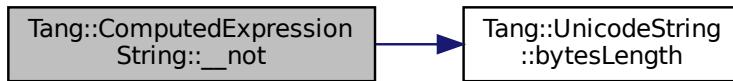
Compute the logical not of this value.

Returns

The result of the operation.

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

Here is the call graph for this function:



5.40.3.17 `__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.

5.40.3.18 `__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

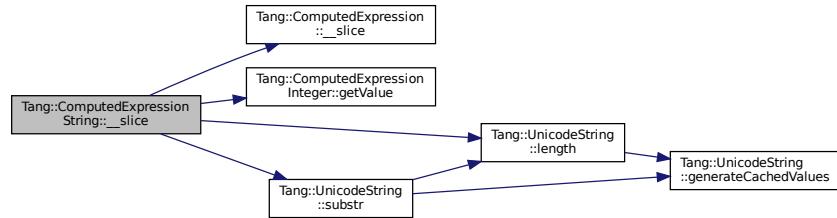
<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

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

Here is the call graph for this function:



5.40.3.19 __string()

```
GarbageCollected ComputedExpressionString::__string () const [override], [virtual]
```

Perform a type cast to string.

Returns

The result of the the operation.

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

5.40.3.20 __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](#).

5.40.3.21 `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](#).

5.40.3.22 `getMethods()`

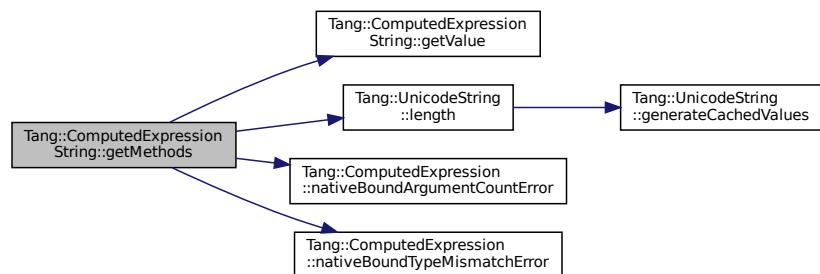
```
NativeBoundFunctionMap ComputedExpressionString::getMethods ( ) [static]
```

Return the member functions implemented for this particular expression type.

Returns

The member functions implemented.

Here is the call graph for this function:



5.40.3.23 `getValue()`

```
UnicodeString ComputedExpressionString::getValue ( ) const
```

Return the string value that is stored in this object.

Returns

The string value.

5.40.3.24 `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

<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:



5.40.3.25 `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](#).

5.40.3.26 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.

5.40.3.27 is_equal() [4/6]

```
bool ComputedExpressionString::is_equal (
    const 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](#).

5.40.3.28 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

val	The value to compare against.
-----	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.40.3.29 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

val	The value to compare against.
-----	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.40.3.30 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](#).

5.40.3.31 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`.

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

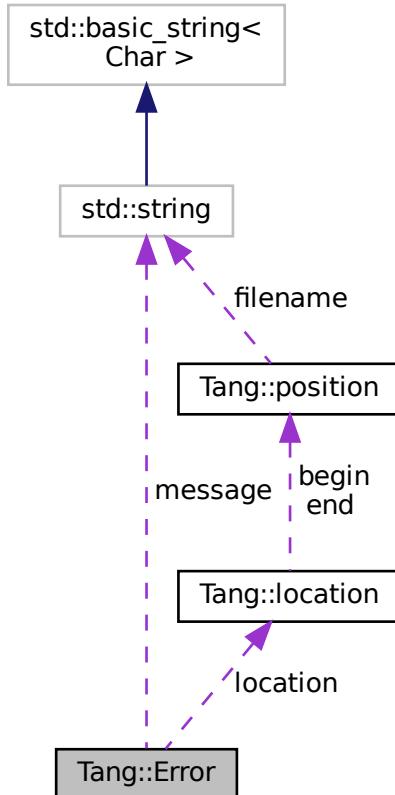
- `include/computedExpressionString.hpp`
- `src/computedExpressionString.cpp`

5.41 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.

5.41.1 Detailed Description

The `Error` class is used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error.

5.41.2 Constructor & Destructor Documentation

5.41.2.1 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.
----------------------	--------------------------------

5.41.2.2 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

<i>message</i>	The error message as a string.
<i>location</i>	The location of the error.

5.41.3 Friends And Related Function Documentation

5.41.3.1 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:

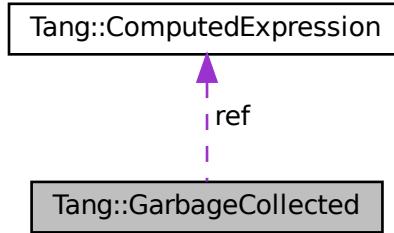
- [include/error.hpp](#)
- [src/error.cpp](#)

5.42 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`
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.
- `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.

5.42.1 Detailed Description

A container that acts as a resource-counting garbage collector for the specified type.

Uses the `SingletonObjectPool` to create and recycle object memory. The container is not thread-safe.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 `GarbageCollected()` [1/3]

```
GarbageCollected::GarbageCollected (
    const GarbageCollected & other )
```

Copy Constructor.

Parameters

<i>The</i>	other <code>GarbageCollected</code> object to copy.
------------	---

5.42.2.2 `GarbageCollected()` [2/3]

```
GarbageCollected::GarbageCollected (
    GarbageCollected && other )
```

Move Constructor.

Parameters

<i>The</i>	other <code>GarbageCollected</code> object to move.
------------	---

5.42.2.3 `~GarbageCollected()`

```
GarbageCollected::~GarbageCollected ( )
```

Destructor.

Clean up the tracked object, if appropriate.

5.42.2.4 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.
-----------------	---

5.42.3 Member Function Documentation

5.42.3.1 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.

5.42.3.2 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:



5.42.3.3 `makeCopy()`

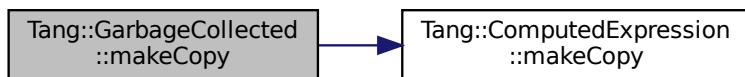
[GarbageCollected](#) `GarbageCollected::makeCopy () const`

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:



5.42.3.4 `operator"!"()`

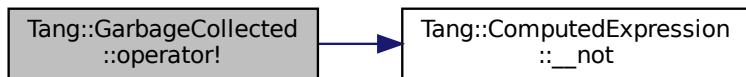
[GarbageCollected](#) `GarbageCollected::operator! () const`

Perform a logical not on the [GarbageCollected](#) value.

Returns

The result of the operation.

Here is the call graph for this function:



5.42.3.5 operator"!=()

```
GarbageCollected GarbageCollected::operator!= ( const GarbageCollected & rhs ) const
```

Perform a != between two [GarbageCollected](#) values.

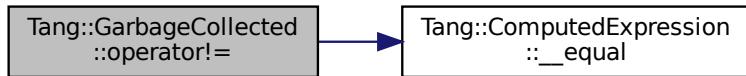
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.42.3.6 operator%()

```
GarbageCollected GarbageCollected::operator% ( const GarbageCollected & rhs ) const
```

Perform a modulo between two [GarbageCollected](#) values.

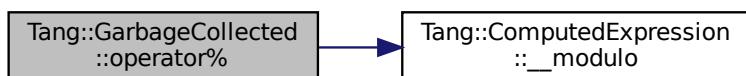
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:

**5.42.3.7 operator*() [1/2]**

`ComputedExpression & GarbageCollected::operator* () const`

Access the tracked object.

Returns

A reference to the tracked object.

5.42.3.8 operator*() [2/2]

`GarbageCollected GarbageCollected::operator* (const GarbageCollected & rhs) const`

Perform a multiplication between two `GarbageCollected` values.

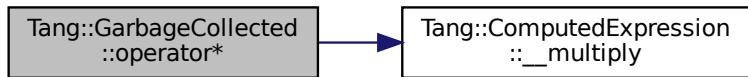
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.42.3.9 operator+()

```
GarbageCollected GarbageCollected::operator+ (\n    const GarbageCollected & rhs ) const
```

Perform an addition between two `GarbageCollected` values.

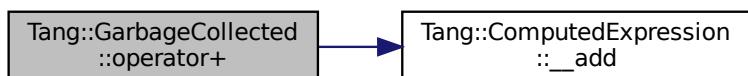
Parameters

<code>rhs</code>	The right hand side operand.
------------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.42.3.10 operator-() [1/2]

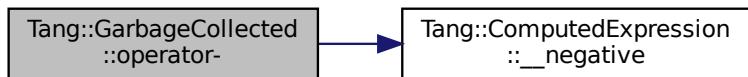
```
GarbageCollected GarbageCollected::operator- ( ) const
```

Perform a negation on the `GarbageCollected` value.

Returns

The result of the operation.

Here is the call graph for this function:

**5.42.3.11 operator-() [2/2]**

```
GarbageCollected GarbageCollected::operator- (
    const GarbageCollected & rhs ) const
```

Perform a subtraction between two `GarbageCollected` values.

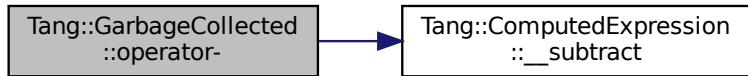
Parameters

<code>rhs</code>	The right hand side operand.
------------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:

**5.42.3.12 operator->()**

```
ComputedExpression * GarbageCollected::operator-> ( ) const
```

Access the tracked object as a pointer.

Returns

A pointer to the tracked object.

5.42.3.13 operator/()

```
GarbageCollected GarbageCollected::operator/ (
    const GarbageCollected & rhs ) const
```

Perform a division between two **GarbageCollected** values.

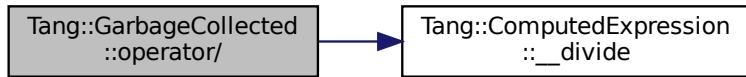
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:

**5.42.3.14 operator<()**

```
GarbageCollected GarbageCollected::operator< (
    const GarbageCollected & rhs ) const
```

Perform a < between two **GarbageCollected** values.

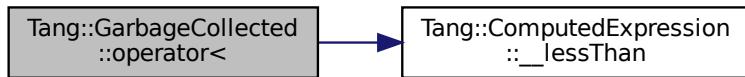
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.42.3.15 operator<=()

```
GarbageCollected GarbageCollected::operator<= (\n    const GarbageCollected & rhs ) const
```

Perform a \leq between two `GarbageCollected` values.

Parameters

<code>rhs</code>	The right hand side operand.
------------------	------------------------------

Returns

The result of the operation.

5.42.3.16 operator=() [1/2]

```
GarbageCollected & GarbageCollected::operator= (\n    const GarbageCollected & other )
```

Copy Assignment.

Parameters

<code>The</code>	other <code>GarbageCollected</code> object.
------------------	---

5.42.3.17 operator=() [2/2]

```
GarbageCollected & GarbageCollected::operator= (\n    GarbageCollected && other )
```

Move Assignment.

Parameters

<i>The</i>	other GarbageCollected object.
------------	--

5.42.3.18 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.

5.42.3.19 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.

5.42.3.20 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.

5.42.3.21 **operator==()** [4/8]

```
GarbageCollected GarbageCollected::operator== (
    const GarbageCollected & rhs ) const
```

Perform a == between two [GarbageCollected](#) values.

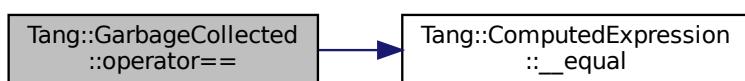
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:

5.42.3.22 **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.

5.42.3.23 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.

5.42.3.24 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.

5.42.3.25 operator==() [8/8]

```
bool GarbageCollected::operator== (
    const Tang::integer_t & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

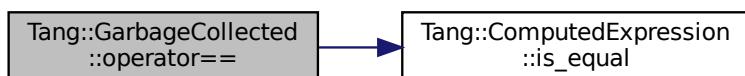
Parameters

<code>val</code>	The value to compare the tracked object against.
------------------	--

Returns

True if they are equal, false otherwise.

Here is the call graph for this function:



5.42.3.26 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.

5.42.3.27 operator>=()

```
GarbageCollected GarbageCollected::operator>= (
    const GarbageCollected & rhs ) const
```

Perform a `>=` between two `GarbageCollected` values.

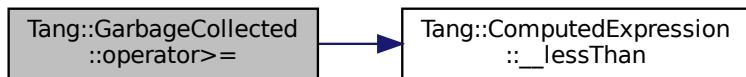
Parameters

<code>rhs</code>	The right hand side operand.
------------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.42.4 Friends And Related Function Documentation

5.42.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & out,
    const GarbageCollected & gc ) [friend]
```

Add friendly output.

Parameters

<i>out</i>	The output stream.
<i>gc</i>	The GarbageCollected value.

Returns

The output stream.

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

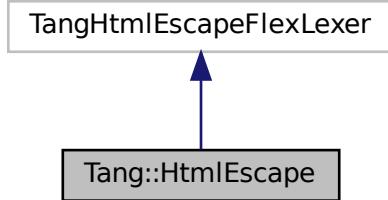
- [include/garbageCollected.hpp](#)
- [src/garbageCollected.cpp](#)

5.43 Tang::HtmlEscape Class Reference

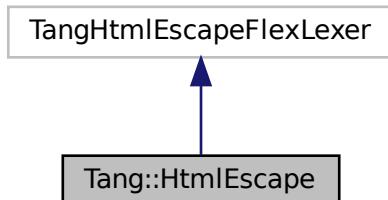
The Flex lexer class for the main Tang language.

```
#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.

5.43.1 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.

5.43.2 Constructor & Destructor Documentation

5.43.2.1 `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)

5.43.3 Member Function Documentation

5.43.3.1 `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:

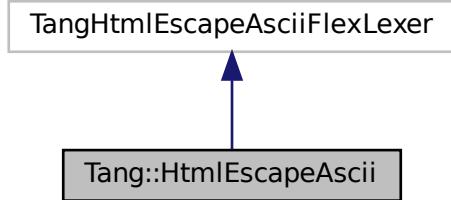
- [include/htmlEscape.hpp](#)

5.44 `Tang::HtmlEscapeAscii` Class Reference

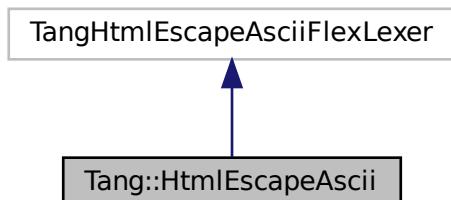
The Flex lexer class for the main Tang language.

```
#include <htmlEscapeAscii.hpp>
```

Inheritance diagram for Tang::HtmlEscapeAscii:



Collaboration diagram for Tang::HtmlEscapeAscii:



Public Member Functions

- [`HtmlEscapeAscii` \(std::istream &arg_yyin, std::ostream &arg_yyout\)](#)
The constructor for the Scanner.
- [`virtual std::string get_next_token \(\)`](#)
Extract the next token from the input string.

5.44.1 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.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 `HtmlEscapeAscii()`

```
Tang::HtmlEscapeAscii::HtmlEscapeAscii (
    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)

5.44.3 Member Function Documentation

5.44.3.1 `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:

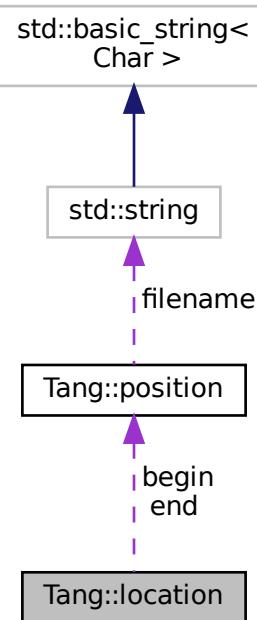
- [include/htmlEscapeAscii.hpp](#)

5.45 `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.

5.45.1 Detailed Description

Two points in a source file.

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

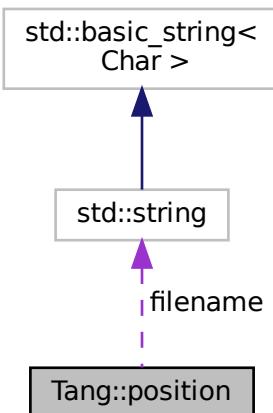
- build/generated/[location.hh](#)

5.46 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 *f=((void *) 0), counter_type l=1, counter_type c=1)`
`Construct a position.`
- `void initialize (filename_type *fn=((void *) 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).`

5.46.1 Detailed Description

A point in a source file.

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

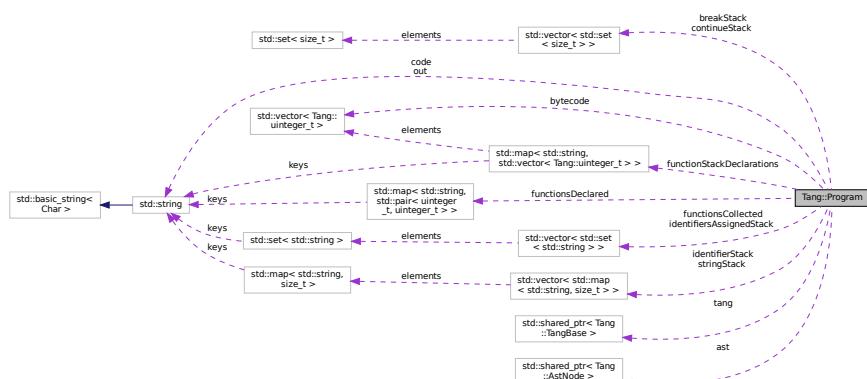
- `build/generated/location.hh`

5.47 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.

- `Program` & `execute` ()

Execute the program's Bytecode, and return the current `Program` object.

- 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 `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`)

Add a string to the environment.

- const std::map< std::string, size_t > & `getStrings` () const

Get the string map of the current environment.

- void `pushBreakStack` ()

Increase the break environment stack, so that we can handle nested break-supporting structures.

- 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::string `out`

The output of the program, resulting from the program execution.
- 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::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::string, 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.
- 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` execution.

5.47.1 Detailed Description

Represents a compiled script or template that may be executed.

5.47.2 Member Enumeration Documentation

5.47.2.1 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.

5.47.3 Constructor & Destructor Documentation

5.47.3.1 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.

5.47.4 Member Function Documentation

5.47.4.1 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.
-----------------	---

5.47.4.2 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.

5.47.4.3 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.
-----------------	--

5.47.4.4 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.

5.47.4.5 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.
-------------	----------------------

5.47.4.6 addString()

```
void Program::addString (
    const std::string & name )
```

Add a string 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.

5.47.4.7 dumpBytecode()

```
string Program::dumpBytecode ( ) const
```

Get the Opcodes of the compiled program, formatted like Assembly.

Returns

A string containing the Opcode representation.

5.47.4.8 execute()

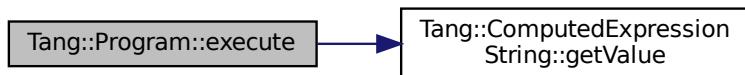
```
Program & Program::execute ( )
```

Execute the program's Bytecode, and return the current [Program](#) object.

Returns

The current [Program](#) object.

Here is the call graph for this function:



5.47.4.9 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.

5.47.4.10 getBytecode()

```
const Bytecode & Program::getBytecode ( )
```

Get the Bytecode vector.

Returns

The Bytecode vector.

5.47.4.11 `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.

5.47.4.12 `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.

5.47.4.13 `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.

5.47.4.14 `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.

5.47.4.15 `getStrings()`

```
const map< string, 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.

5.47.4.16 `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:



5.47.4.17 `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:



5.47.4.18 pushEnvironment()

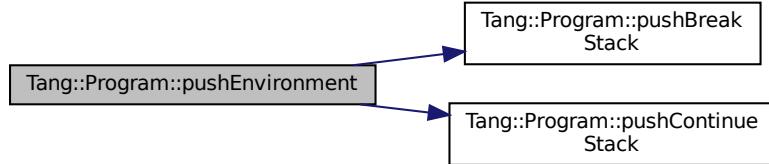
```
void Program::pushEnvironment (
    const std::shared_ptr< AstNode > & ast )
```

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:



5.47.4.19 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.

5.47.4.20 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.

5.47.5 Member Data Documentation**5.47.5.1 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:

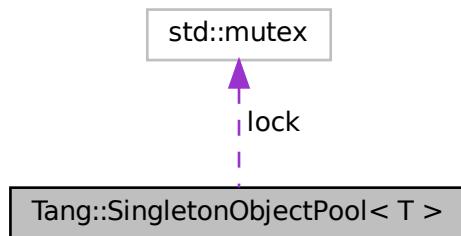
- [include/program.hpp](#)
- [src/program-dumpBytecode.cpp](#)
- [src/program-execute.cpp](#)
- [src/program.cpp](#)

5.48 `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.

5.48.1 Detailed Description

```
template<class T>
class Tang::SingletonObjectPool< T >
```

A thread-safe, singleton object pool of the designated type.

5.48.2 Member Function Documentation

5.48.2.1 `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`.

5.48.2.2 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.

5.48.2.3 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.
------------	---------------------------------

5.48.3 Member Data Documentation

5.48.3.1 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.

5.48.3.2 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:

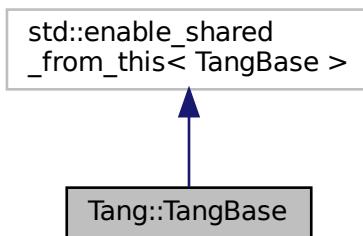
- [include/SingletonObjectPool.hpp](#)

5.49 Tang::TangBase Class Reference

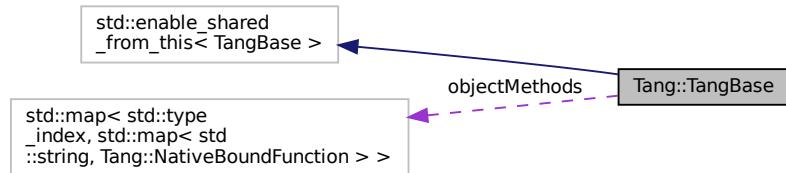
The base class for the Tang programming language.

```
#include <tangBase.hpp>
```

Inheritance diagram for Tang::TangBase:



Collaboration diagram for Tang::TangBase:



Public Member Functions

- [Program compileScript \(std::string script\)](#)
Compile the provided source code as a script and return a [Program](#).
- [TangBase \(\)](#)
The constructor.
- [map< std::type_index, std::map< std::string, Tang::NativeBoundFunction > > & getObjectMethods \(\)](#)
Get the object methods 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

- `map< std::type_index, std::map< std::string, Tang::NativeBoundFunction > > objectMethods`
Store the available object methods.

5.49.1 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).

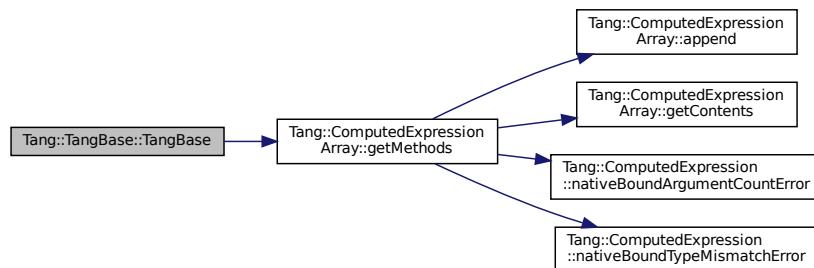
5.49.2 Constructor & Destructor Documentation

5.49.2.1 [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:



5.49.3 Member Function Documentation

5.49.3.1 [compileScript\(\)](#)

```
Program TangBase::compileScript (
    std::string script )
```

Compile the provided source code 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.

5.49.3.2 `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:

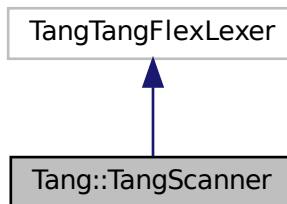
- [include/tangBase.hpp](#)
- [src/tangBase.cpp](#)

5.50 Tang::TangScanner Class Reference

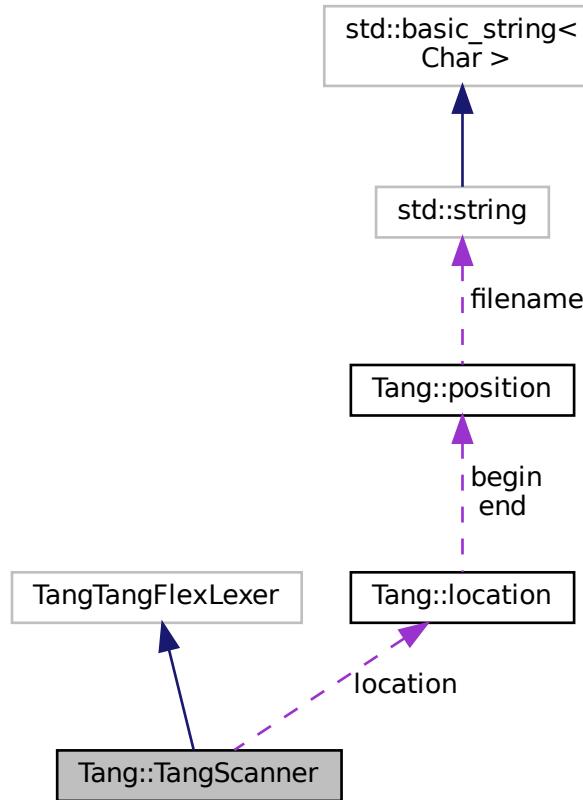
The Flex lexer class for the main Tang language.

```
#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.

Private Attributes

- `Tang::location location`
The location information of the token that is identified.

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 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

<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 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:

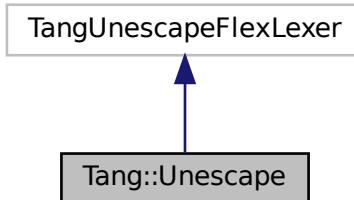
- [include/tangScanner.hpp](#)

5.51 Tang::Unescape Class Reference

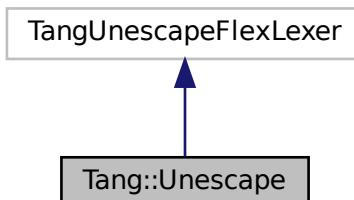
The Flex lexer class for the main Tang language.

```
#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.

5.51.1 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 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

<code>arg_yyin</code>	The input stream to be tokenized
<code>arg_yyout</code>	The output stream (not currently used)

5.51.3 Member Function Documentation

5.51.3.1 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:

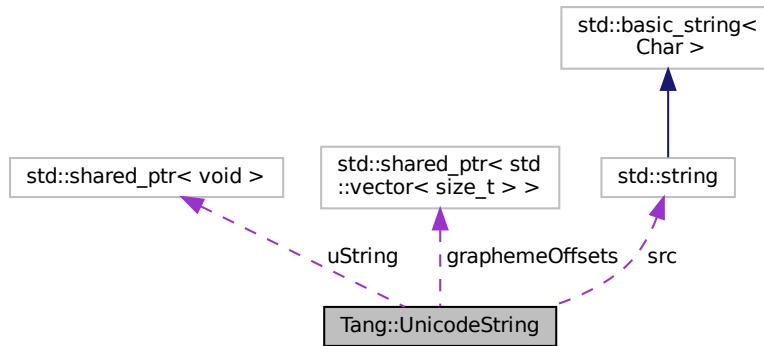
- [include/unescape.hpp](#)

5.52 Tang::UnicodeString Class Reference

Represents a UTF-8 encoded string that is Unicode-aware.

```
#include <unicodeString.hpp>
```

Collaboration diagram for Tang::UnicodeString:



Public Member Functions

- **UnicodeString** (const std::string &**src**)
Construct a [Tang::UnicodeString](#) object, which acts as the interface to the ICU library.
- 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 [UnicodeString](#)s.
- [UnicodeString](#) **operator+** (const [UnicodeString](#) &rhs) const
Create a new [UnicodeString](#) that is the concatenation of two [UnicodeString](#)s.
- **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.

Private Member Functions

- void **generateCachedValues ()** const
Calculate cacheable values for the object.

Private Attributes

- std::string `src`
The UTF-8 encoded string.
- 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.

5.52.1 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.

5.52.2 Constructor & Destructor Documentation

5.52.2.1 UnicodeString()

```
UnicodeString::UnicodeString (   
    const std::string & src )
```

Construct a [Tang::UnicodeString](#) object, which acts as the interface to the ICU library.

Parameters

<code>src</code>	A UTF-8 encoded string.
------------------	-------------------------

5.52.3 Member Function Documentation

5.52.3.1 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.

5.52.3.2 `length()`

```
size_t UnicodeString::length ( ) const
```

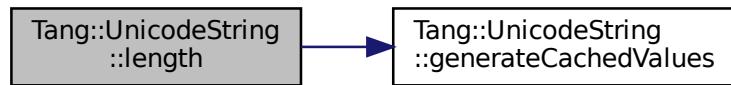
Return the length of the [UnicodeString](#) in graphemes.

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:



5.52.3.3 `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](#).

5.52.3.4 `operator+()`

```
UnicodeString UnicodeString::operator+ (
    const UnicodeString & rhs ) const
```

Create a new [UnicodeString](#) that is the concatenation of two `UnicodeString`s.

Parameters

<i>rhs</i>	The string to append to the current object string.
------------	--

Returns

Returns the result of the concatenation.

5.52.3.5 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.

5.52.3.6 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 two strings are equal.

5.52.3.7 substr()

```
std::string UnicodeString::substr (
    size_t position,
    size_t length ) const
```

Return a Unicode grapheme-aware substring.

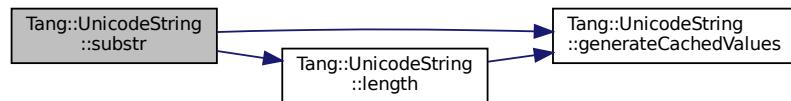
Parameters

<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](#)

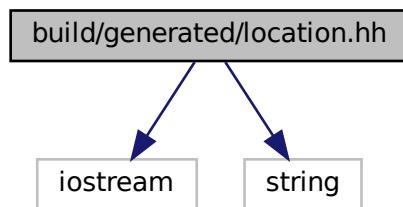
Chapter 6

File Documentation

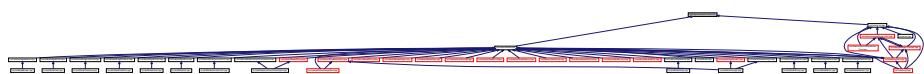
6.1 build/generated/location.hh File Reference

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.

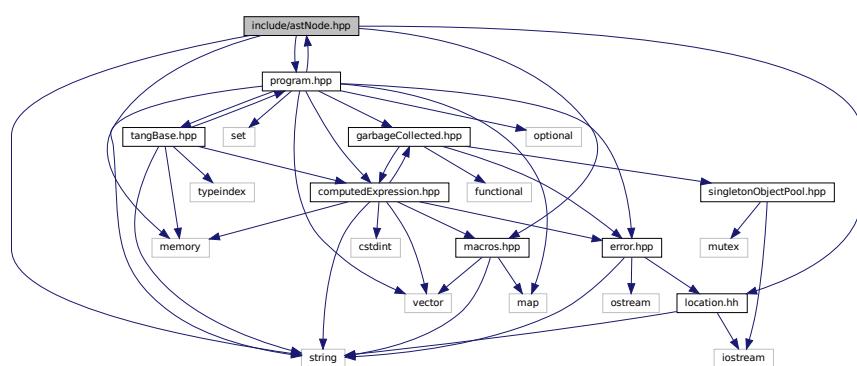
Parameters

<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 "program.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).

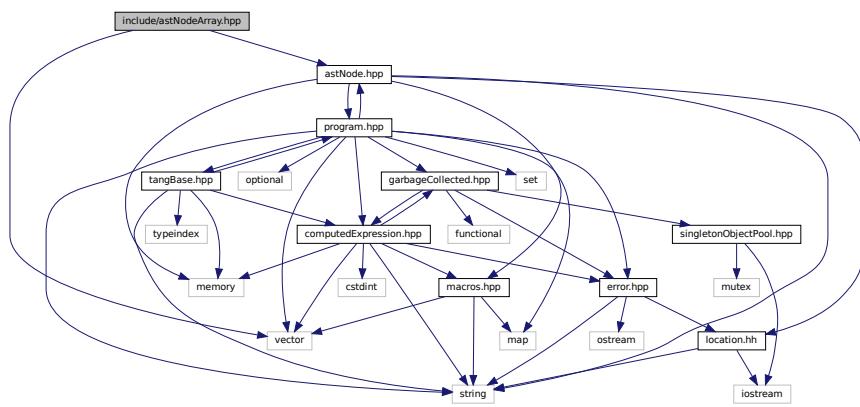
6.2.1 Detailed Description

Declare the [Tang::AstNode](#) base class.

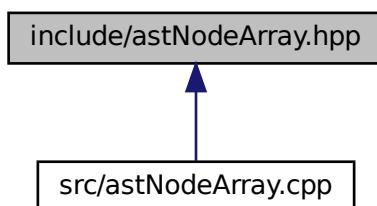
6.3 include/astNodeArray.hpp File Reference

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.

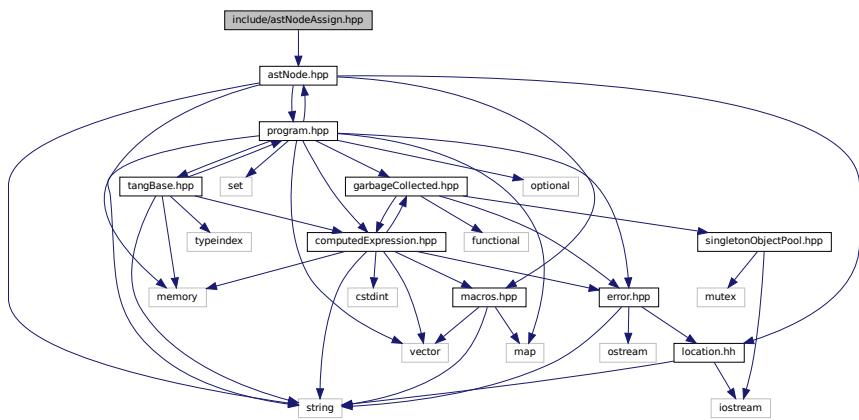
6.3.1 Detailed Description

Declare the `Tang::AstNodeArray` class.

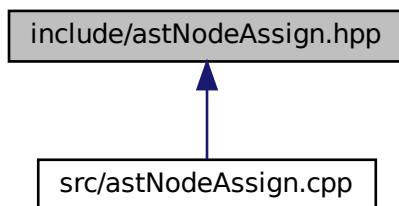
6.4 include/astNodeAssign.hpp File Reference

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.

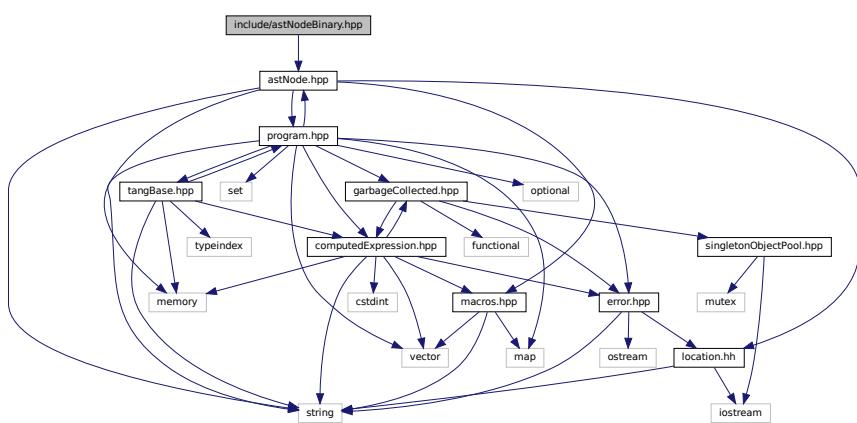
6.4.1 Detailed Description

Declare the [Tang::AstNodeAssign](#) class.

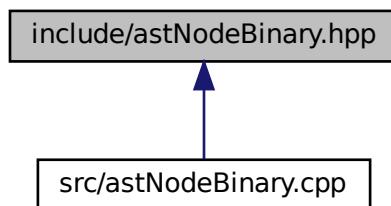
6.5 include/astNodeBinary.hpp File Reference

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.

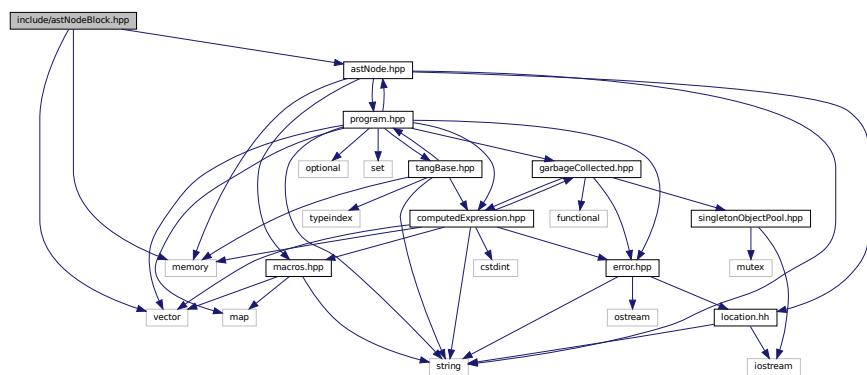
6.5.1 Detailed Description

Declare the [Tang::AstNodeBinary](#) class.

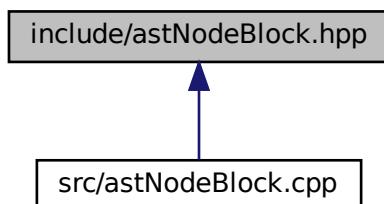
6.6 include/astNodeBlock.hpp File Reference

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

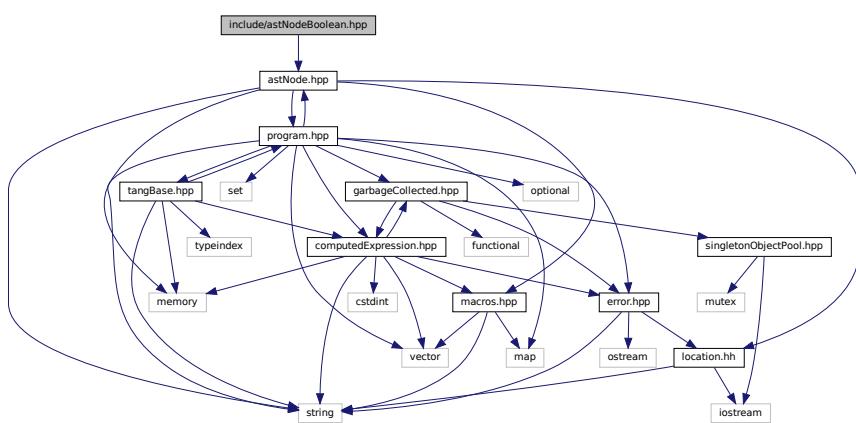
Declare the [Tang::AstNodeBlock](#) class.

6.7 include/astNodeBoolean.hpp File Reference

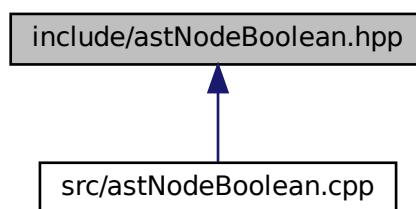
Declare the [Tang::AstNodeBoolean](#) class.

```
#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.

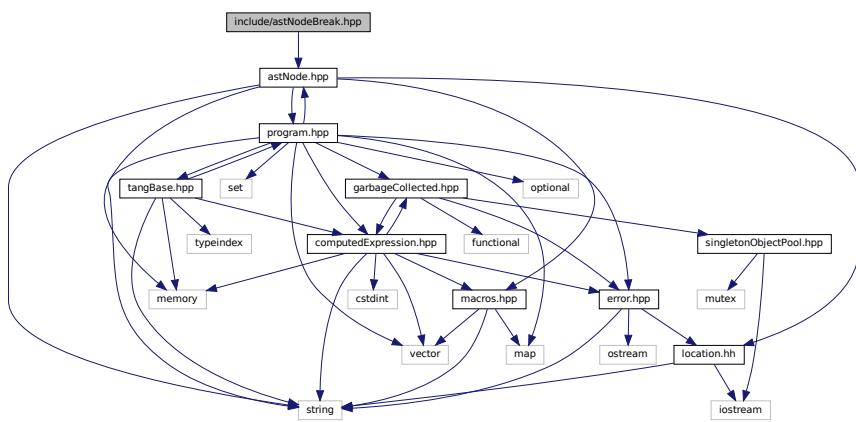
6.7.1 Detailed Description

Declare the [Tang::AstNodeBoolean](#) class.

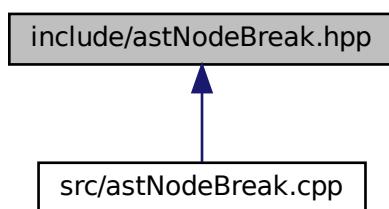
6.8 include/astNodeBreak.hpp File Reference

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

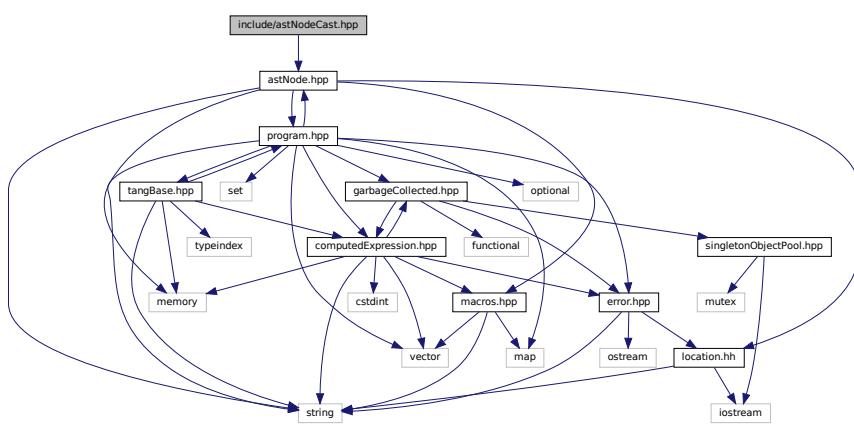
Declare the [Tang::AstNodeBreak](#) class.

6.9 include/astNodeCast.hpp File Reference

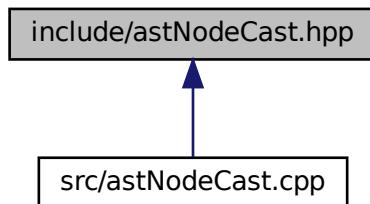
Declare the [Tang::AstNodeCast](#) class.

```
#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.

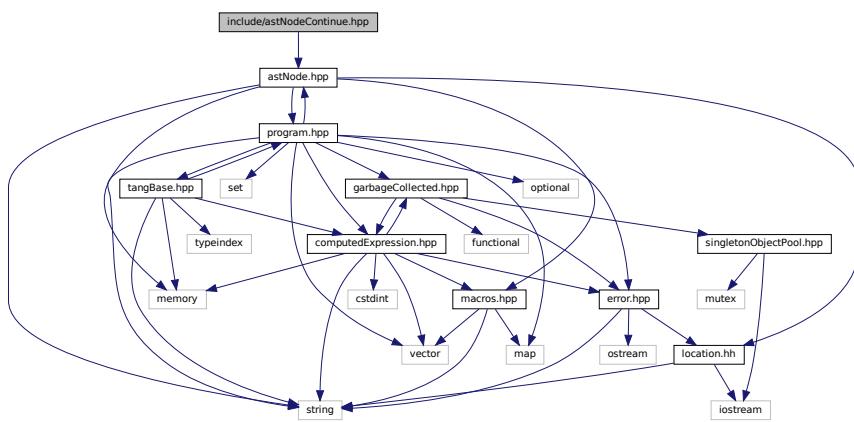
6.9.1 Detailed Description

Declare the [Tang::AstNodeCast](#) class.

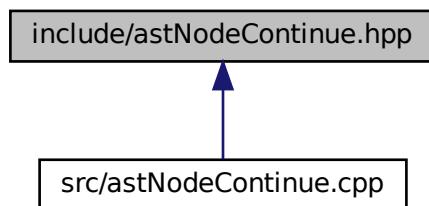
6.10 include/astNodeContinue.hpp File Reference

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.

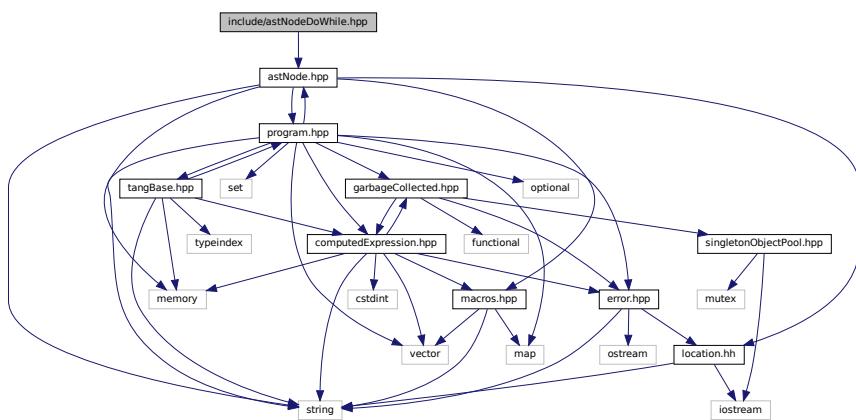
6.10.1 Detailed Description

Declare the [Tang::AstNodeContinue](#) class.

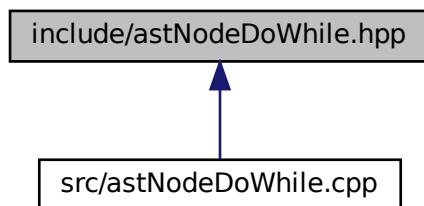
6.11 include/astNodeDoWhile.hpp File Reference

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.

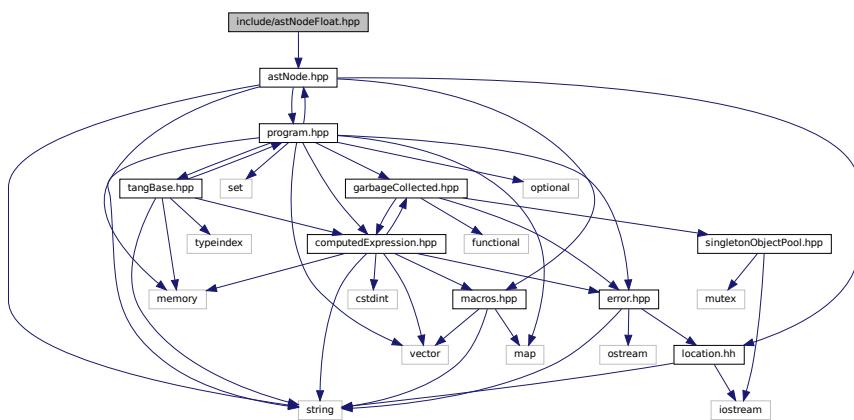
6.11.1 Detailed Description

Declare the `Tang::AstNodeDoWhile` class.

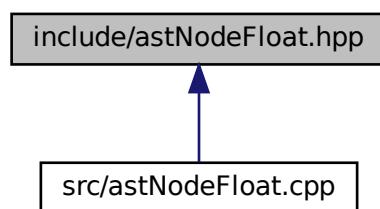
6.12 include/astNodeFloat.hpp File Reference

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

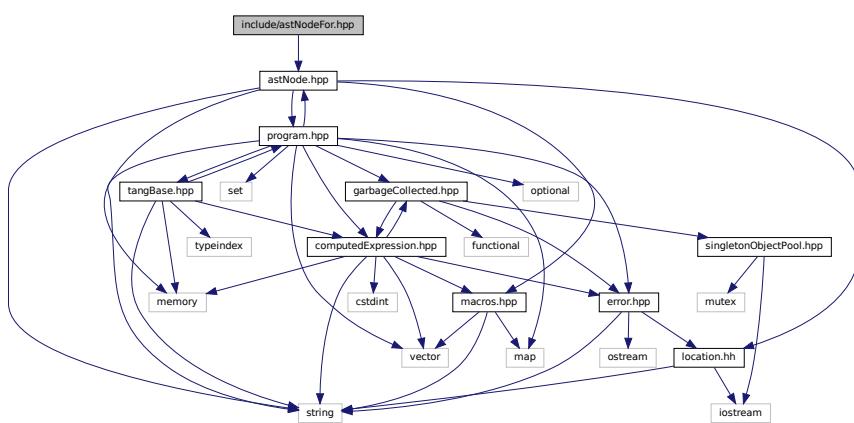
Declare the [Tang::AstNodeFloat](#) class.

6.13 include/astNodeFor.hpp File Reference

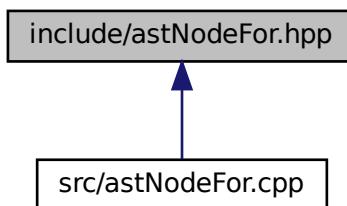
Declare the [Tang::AstNodeFor](#) class.

```
#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.

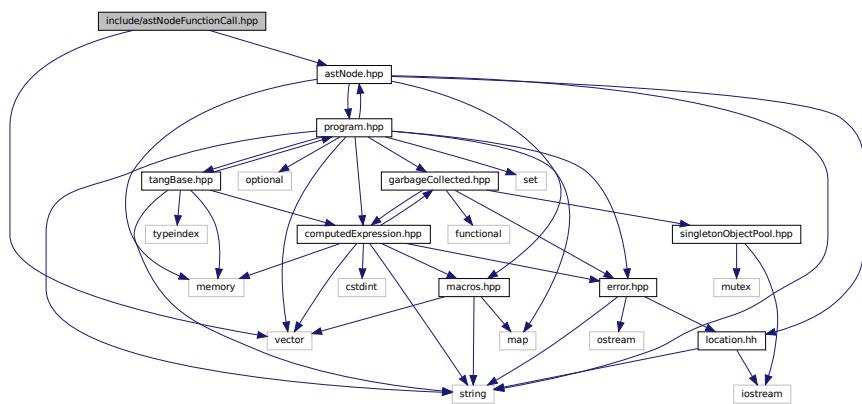
6.13.1 Detailed Description

Declare the [Tang::AstNodeFor](#) class.

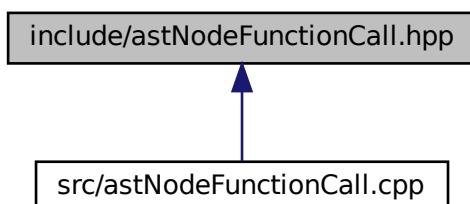
6.14 include/astNodeFunctionCall.hpp File Reference

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.

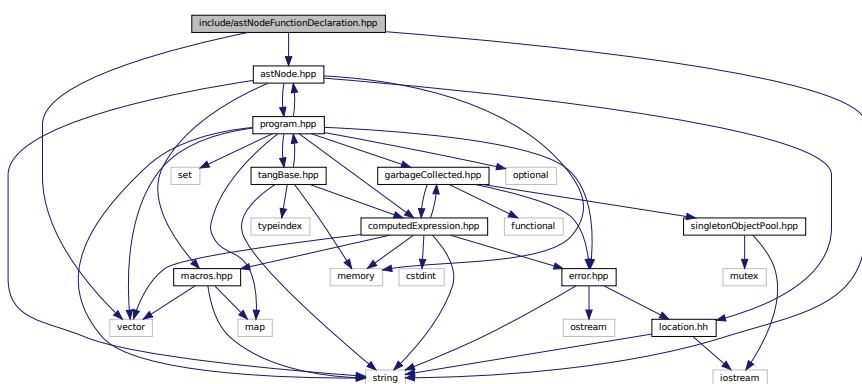
6.14.1 Detailed Description

Declare the [Tang::AstNodeFunctionCall](#) class.

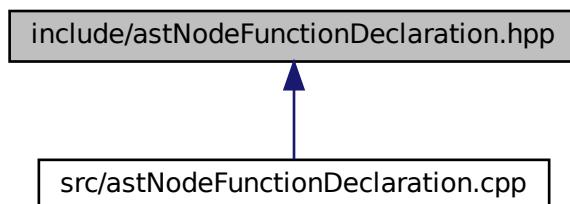
6.15 include/astNodeFunctionDeclaration.hpp File Reference

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.

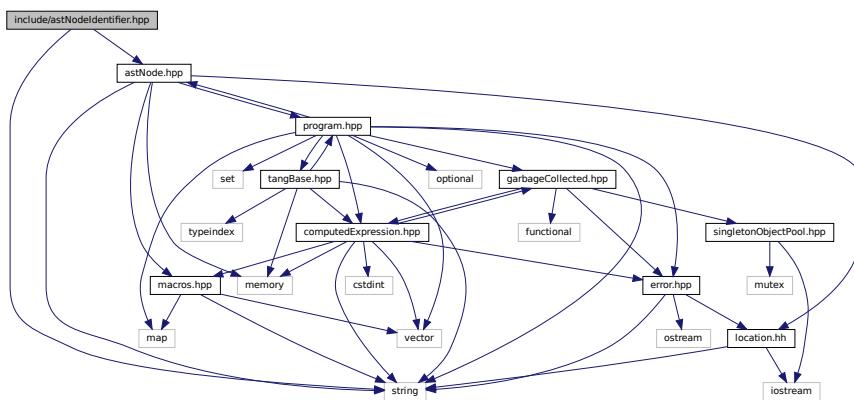
6.15.1 Detailed Description

Declare the [Tang::AstNodeFunctionDeclaration](#) class.

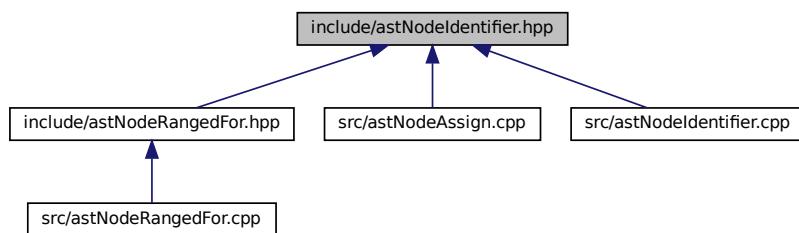
6.16 include/astNodelIdentifier.hpp File Reference

Declare the [Tang::AstNodelIdentifier](#) class.

```
#include <string>
#include "astNode.hpp"
Include dependency graph for astNodelIdentifier.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodelIdentifier](#)
An [AstNode](#) that represents an identifier.

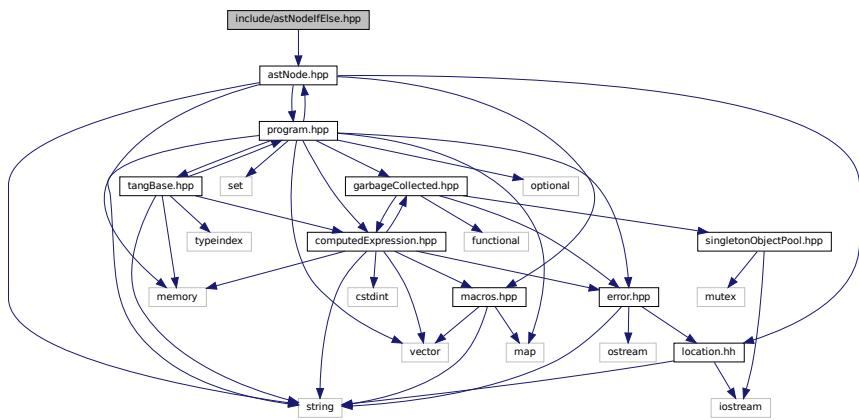
6.16.1 Detailed Description

Declare the [Tang::AstNodeIdentifier](#) class.

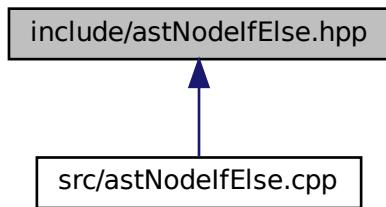
6.17 include/astNodeIfElse.hpp File Reference

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.

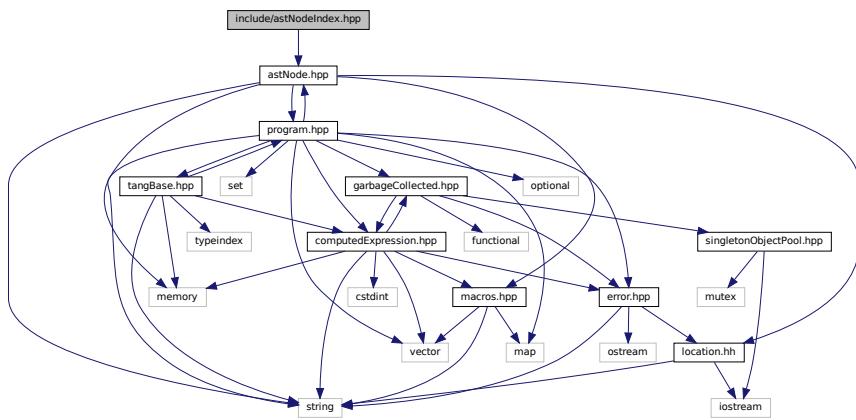
6.17.1 Detailed Description

Declare the [Tang::AstNodeIfElse](#) class.

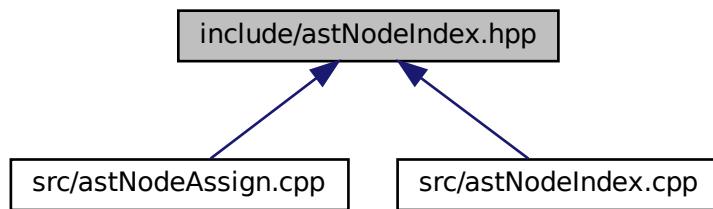
6.18 include/astNodeIndex.hpp File Reference

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.

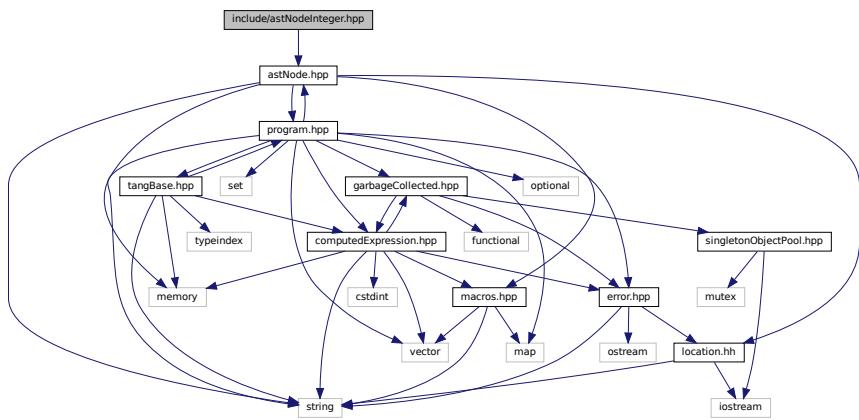
6.18.1 Detailed Description

Declare the [Tang::AstNodeIndex](#) class.

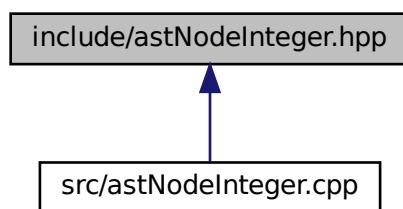
6.19 include/astNodeInteger.hpp File Reference

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.

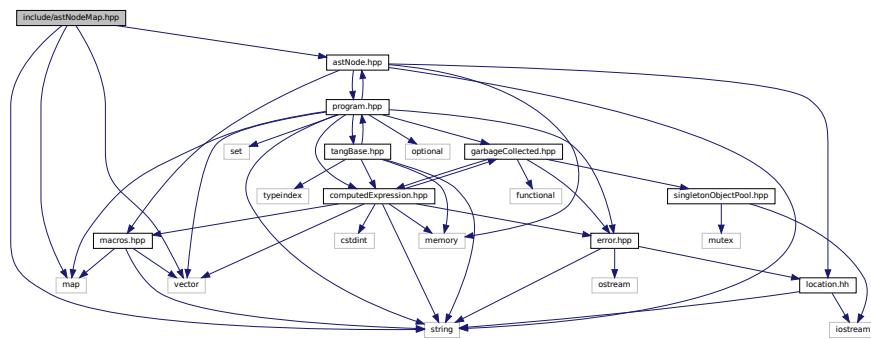
6.19.1 Detailed Description

Declare the [Tang::AstNodeInteger](#) class.

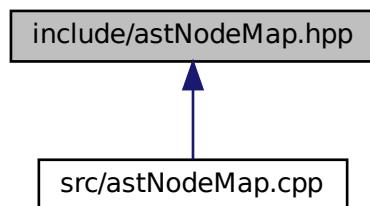
6.20 include/astNodeMap.hpp File Reference

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.

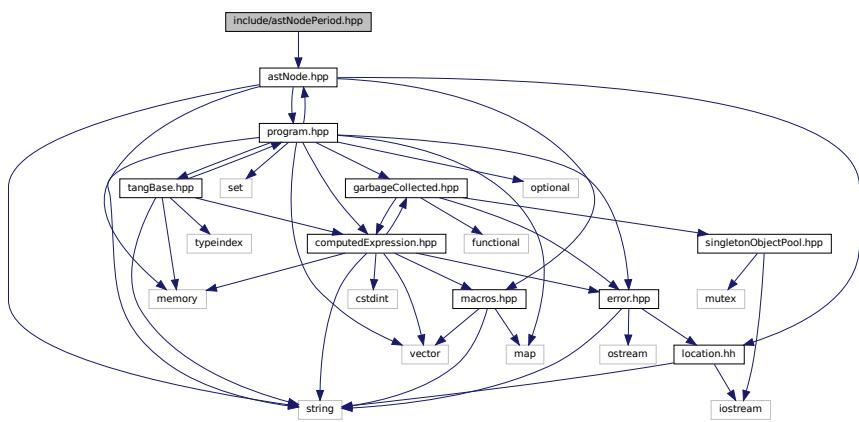
6.20.1 Detailed Description

Declare the [Tang::AstNodeMap](#) class.

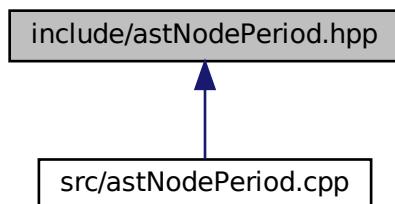
6.21 include/astNodePeriod.hpp File Reference

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.

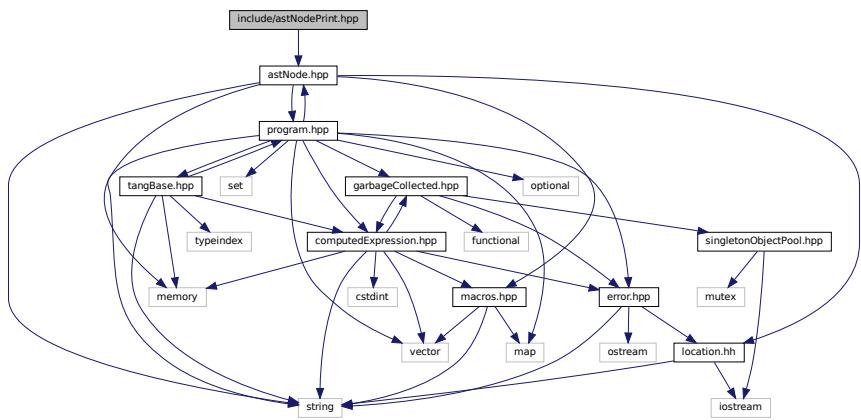
6.21.1 Detailed Description

Declare the [Tang::AstNodePeriod](#) class.

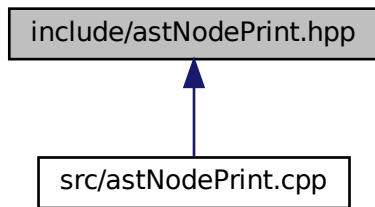
6.22 include/astNodePrint.hpp File Reference

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.

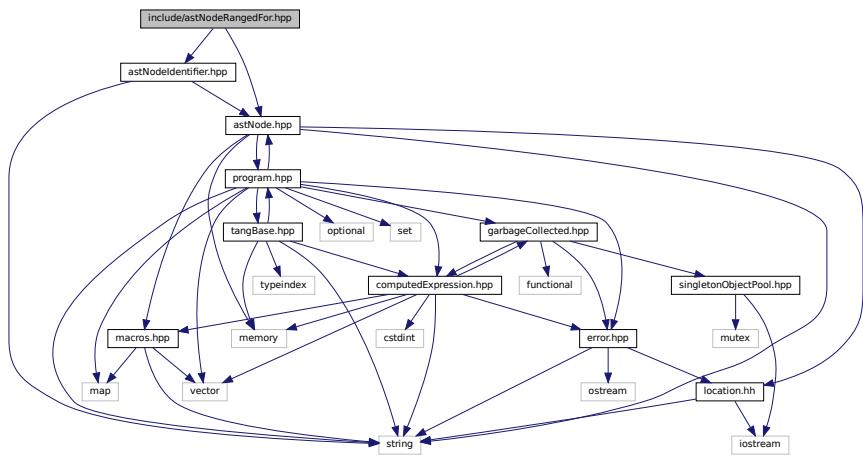
6.22.1 Detailed Description

Declare the `Tang::AstNodePrint` class.

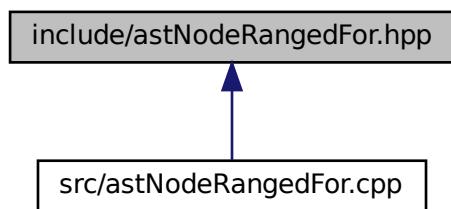
6.23 include/astNodeRangedFor.hpp File Reference

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.

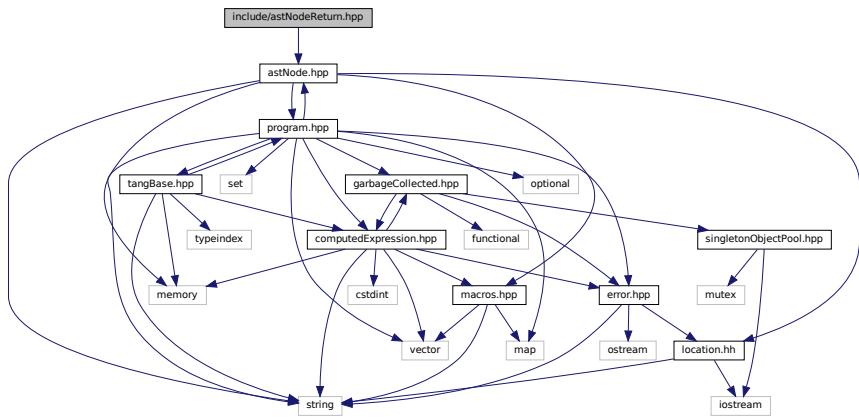
6.23.1 Detailed Description

Declare the `Tang::AstNodeRangedFor` class.

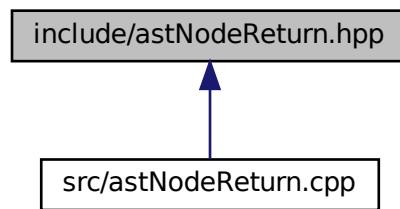
6.24 include/astNodeReturn.hpp File Reference

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.

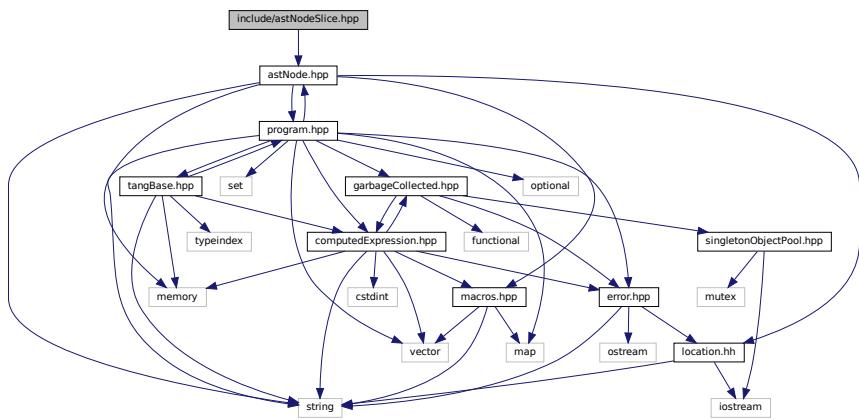
6.24.1 Detailed Description

Declare the [Tang::AstNodeReturn](#) class.

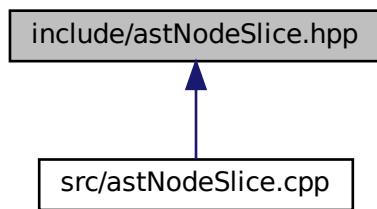
6.25 include/astNodeSlice.hpp File Reference

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.

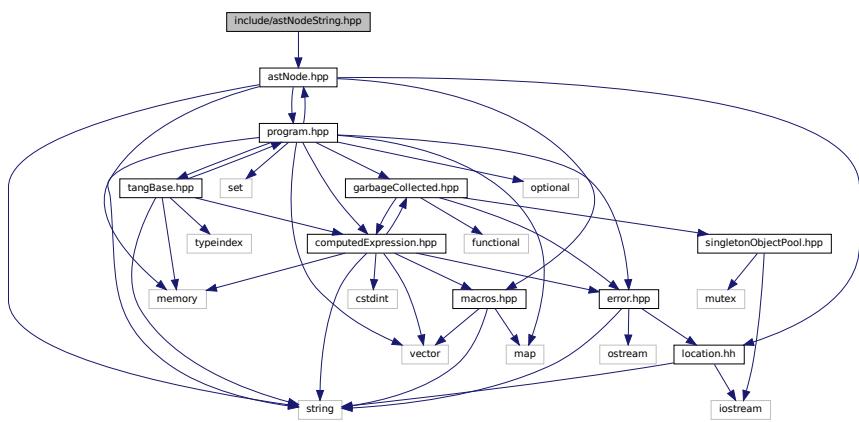
6.25.1 Detailed Description

Declare the [Tang::AstNodeSlice](#) class.

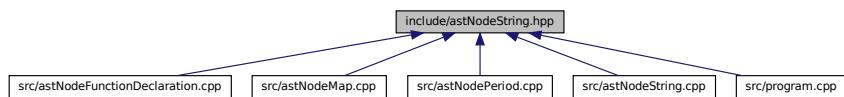
6.26 include/astNodeString.hpp File Reference

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.

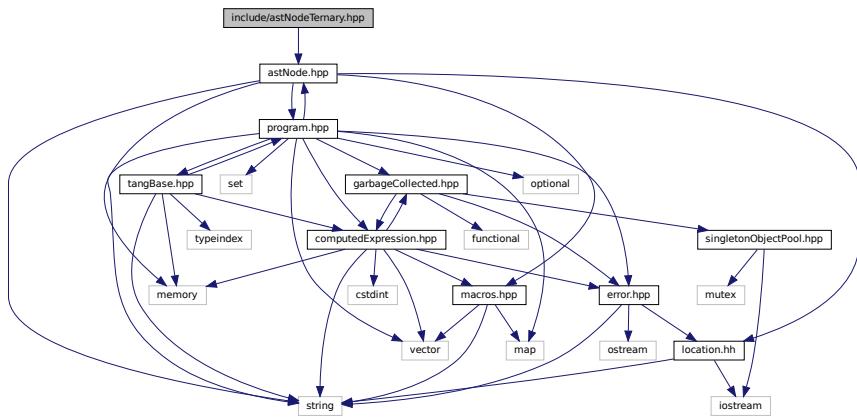
6.26.1 Detailed Description

Declare the [Tang::AstNodeString](#) class.

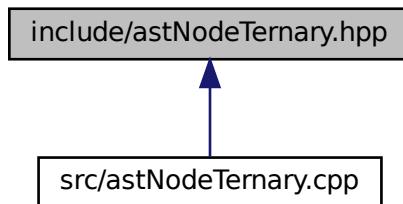
6.27 include/astNodeTernary.hpp File Reference

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.

6.27.1 Detailed Description

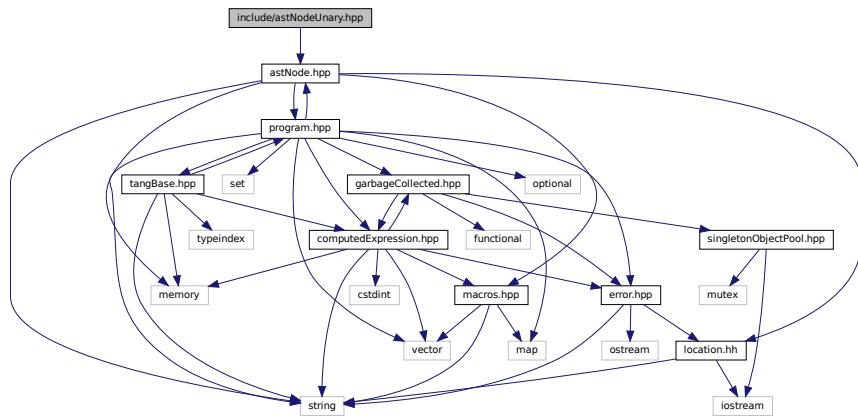
Declare the [Tang::AstNodeTernary](#) class.

6.28 include/astNodeUnary.hpp File Reference

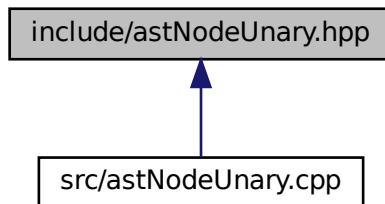
Declare the [Tang::AstNodeUnary](#) class.

```
#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.

6.28.1 Detailed Description

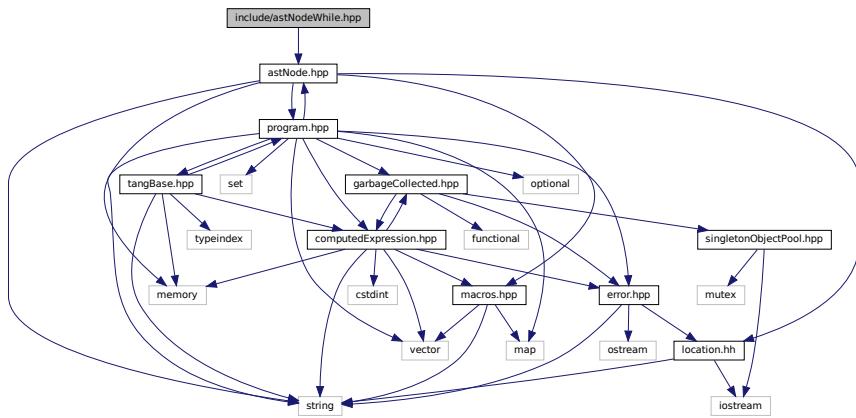
Declare the [Tang::AstNodeUnary](#) class.

6.29 include/astNodeWhile.hpp File Reference

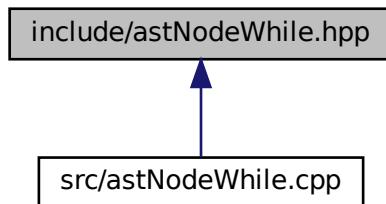
Declare the [Tang::AstNodeWhile](#) class.

```
#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.

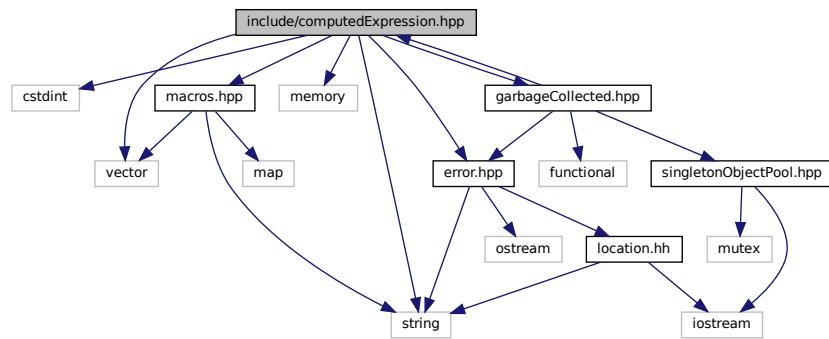
6.29.1 Detailed Description

Declare the [Tang::AstNodeWhile](#) class.

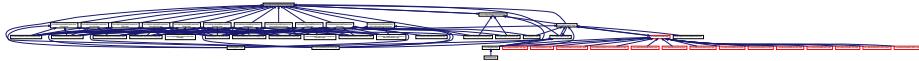
6.30 include/computedExpression.hpp File Reference

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.

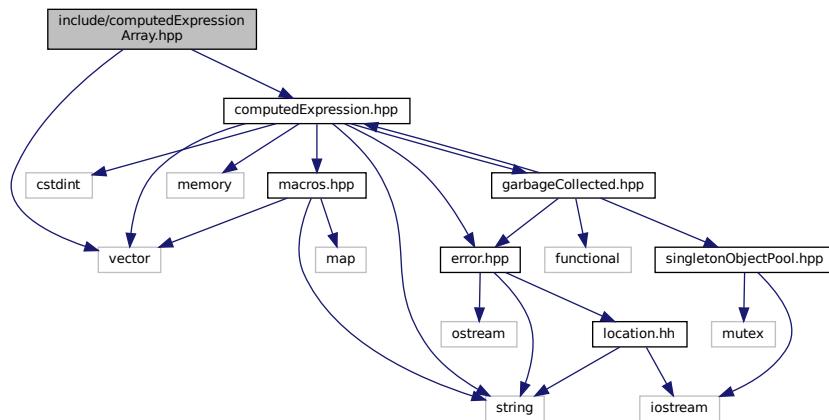
6.30.1 Detailed Description

Declare the [Tang::ComputedExpression](#) base class.

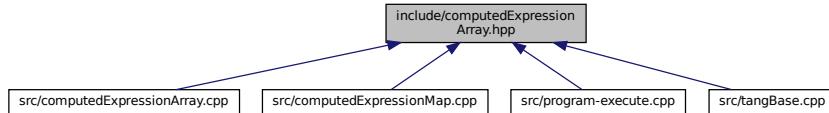
6.31 include/computedExpressionArray.hpp File Reference

Declare the [Tang::ComputedExpressionArray](#) class.

```
#include <vector>
#include "computedExpression.hpp"
Include dependency graph for computedExpressionArray.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionArray](#)
Represents an Array that is the result of a computation.

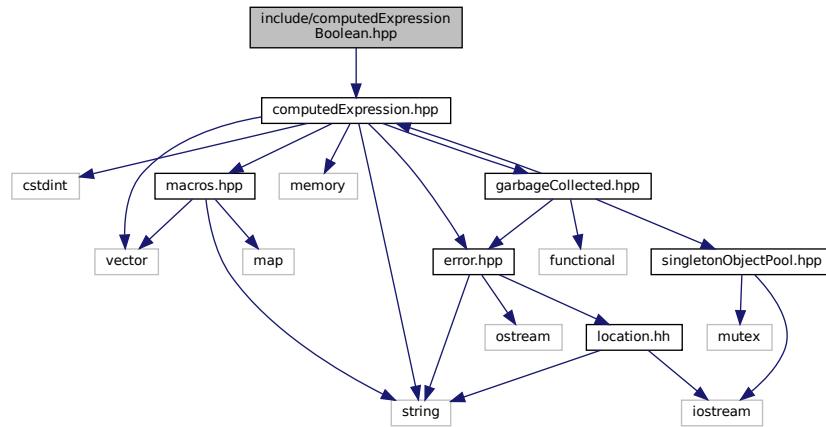
6.31.1 Detailed Description

Declare the [Tang::ComputedExpressionArray](#) class.

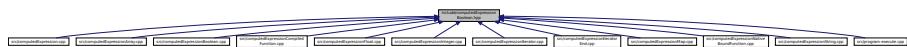
6.32 include/computedExpressionBoolean.hpp File Reference

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.

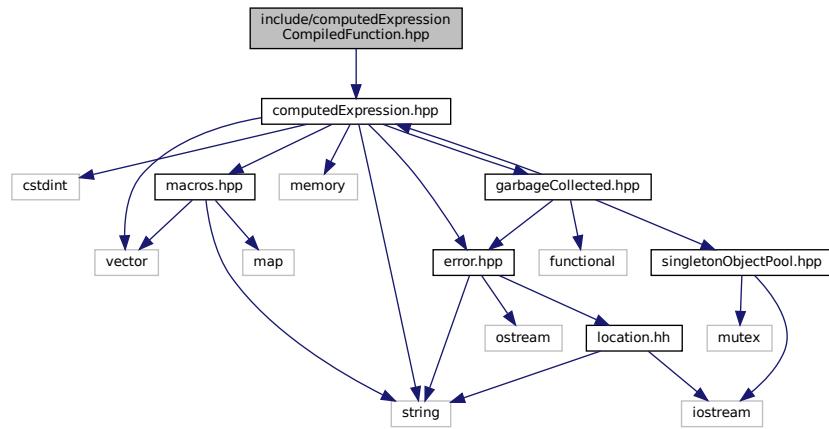
6.32.1 Detailed Description

Declare the [Tang::ComputedExpressionBoolean](#) class.

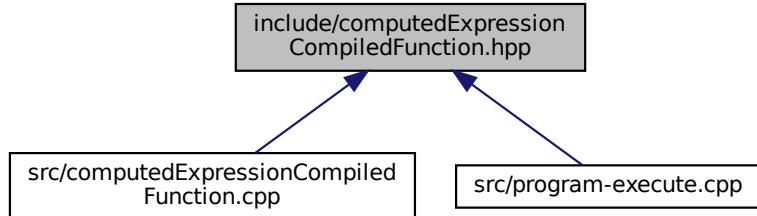
6.33 include/computedExpressionCompiledFunction.hpp File Reference

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.

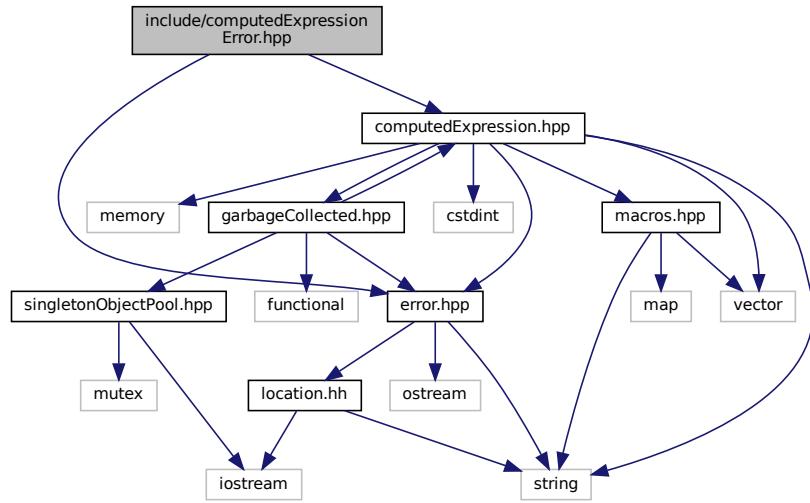
6.33.1 Detailed Description

Declare the [Tang::ComputedExpressionCompiledFunction](#) class.

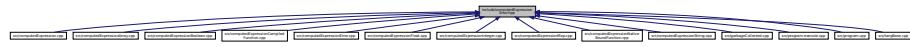
6.34 include/computedExpressionError.hpp File Reference

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.

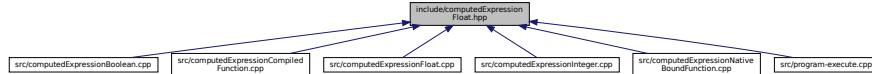
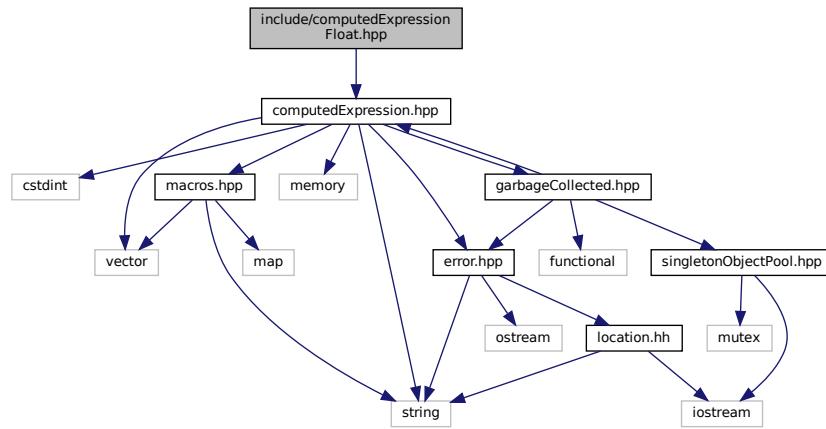
6.34.1 Detailed Description

Declare the [Tang::ComputedExpressionError](#) class.

6.35 include/computedExpressionFloat.hpp File Reference

Declare the [Tang::ComputedExpressionFloat](#) class.

```
#include "computedExpression.hpp"
Include dependency graph for computedExpressionFloat.hpp:
```



Classes

- class [Tang::ComputedExpressionFloat](#)
Represents a Float that is the result of a computation.

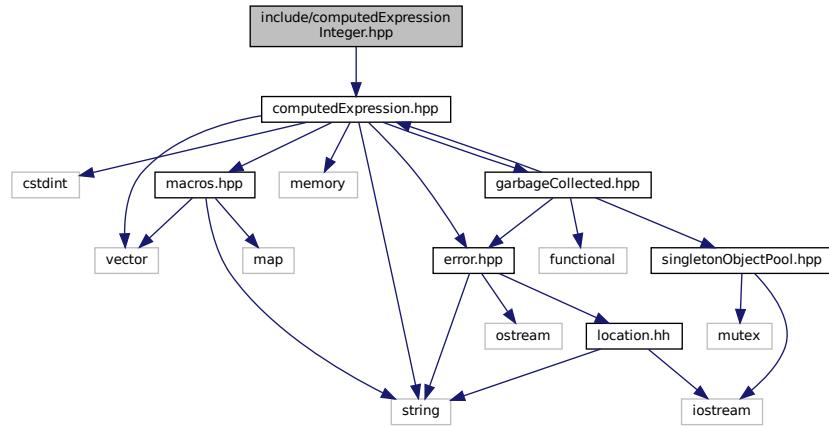
6.35.1 Detailed Description

Declare the [Tang::ComputedExpressionFloat](#) class.

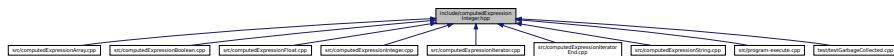
6.36 include/computedExpressionInteger.hpp File Reference

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.

6.36.1 Detailed Description

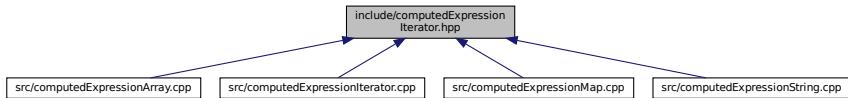
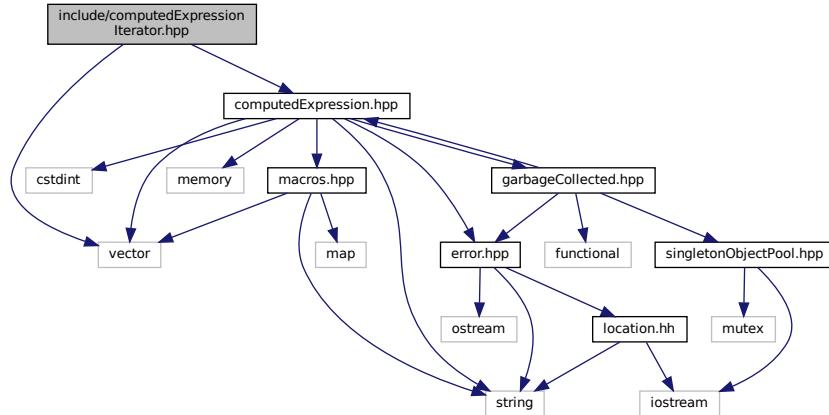
Declare the [Tang::ComputedExpressionInteger](#) class.

6.37 include/computedExpressionIterator.hpp File Reference

Declare the [Tang::ComputedExpressionIterator](#) class.

```
#include <vector>
#include "computedExpression.hpp"
```

Include dependency graph for `computedExpressionIterator.hpp`:



Classes

- class [Tang::ComputedExpressionIterator](#)
Represents an Iterator that is the result of a computation.

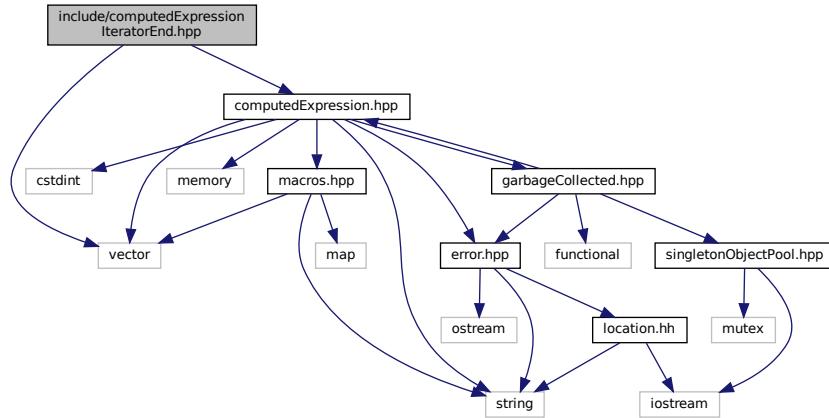
6.37.1 Detailed Description

Declare the [Tang::ComputedExpressionIterator](#) class.

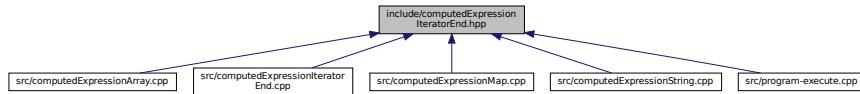
6.38 include/computedExpressionIteratorEnd.hpp File Reference

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.

6.38.1 Detailed Description

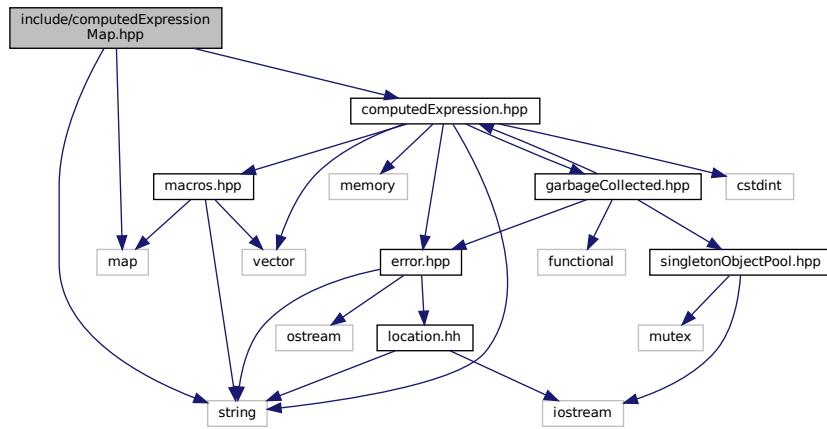
Declare the [Tang::ComputedExpressionIteratorEnd](#) class.

6.39 include/computedExpressionMap.hpp File Reference

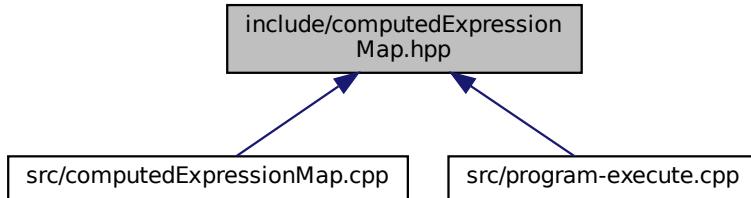
Declare the [Tang::ComputedExpressionMap](#) class.

```
#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.

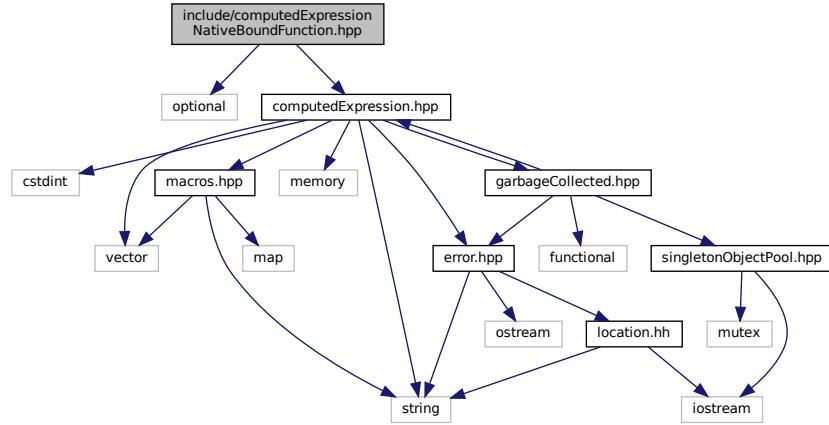
6.39.1 Detailed Description

Declare the [Tang::ComputedExpressionMap](#) class.

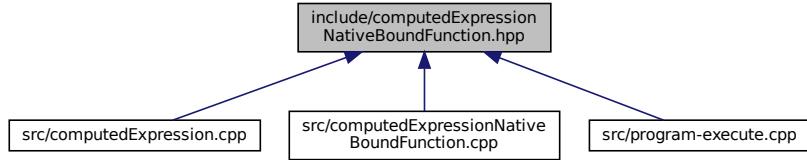
6.40 include/computedExpressionNativeBoundFunction.hpp File Reference

Declare the [Tang::ComputedExpressionNativeBoundFunction](#) class.

```
#include <optional>
#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.

6.40.1 Detailed Description

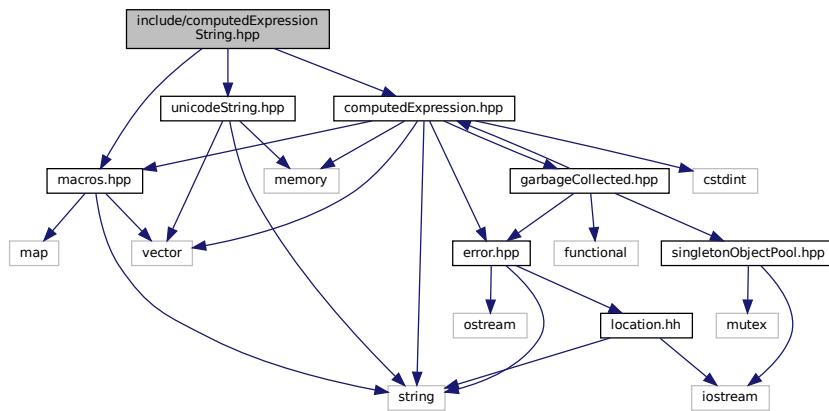
Declare the [Tang::ComputedExpressionNativeBoundFunction](#) class.

6.41 include/computedExpressionString.hpp File Reference

Declare the [Tang::ComputedExpressionString](#) class.

```
#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.

6.41.1 Detailed Description

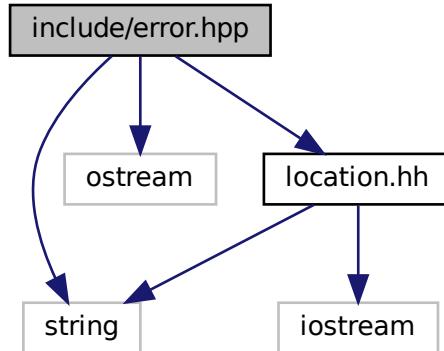
Declare the [Tang::ComputedExpressionString](#) class.

6.42 include/error.hpp File Reference

Declare the [Tang::Error](#) class used to describe syntax and runtime errors.

```
#include <string>
#include <iostream>
```

```
#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.

6.42.1 Detailed Description

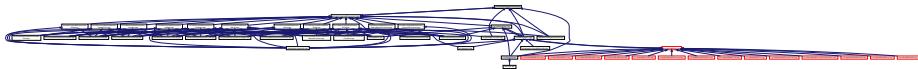
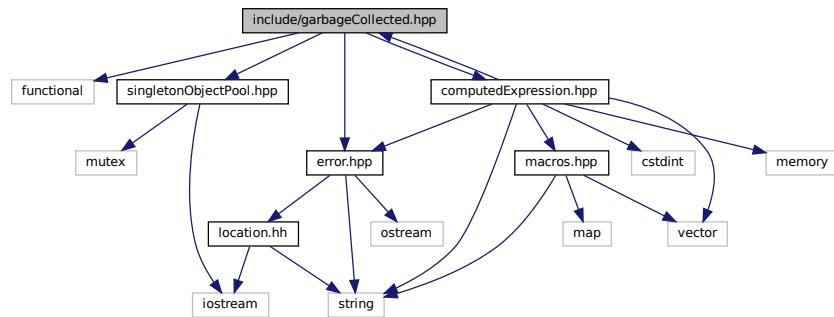
Declare the [Tang::Error](#) class used to describe syntax and runtime errors.

6.43 include/garbageCollected.hpp File Reference

Declare the [Tang::GarbageCollected](#) class.

```
#include <functional>
#include "singletonObjectPool.hpp"
#include "computedExpression.hpp"
```

```
#include "error.hpp"
Include dependency graph for garbageCollected.hpp:
```



Classes

- class [Tang::GarbageCollected](#)
A container that acts as a resource-counting garbage collector for the specified type.

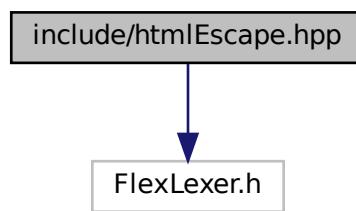
6.43.1 Detailed Description

Declare the [Tang::GarbageCollected](#) class.

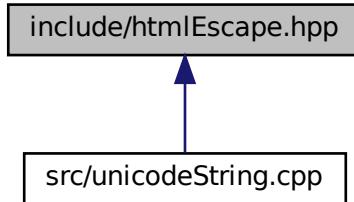
6.44 include/htmlEscape.hpp File Reference

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()`

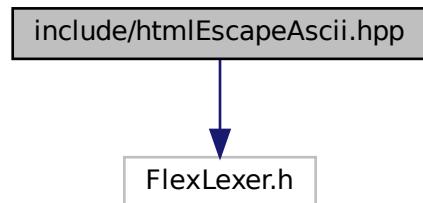
6.44.1 Detailed Description

Declare the [Tang::HtmlEscape](#) used to tokenize a Tang script.

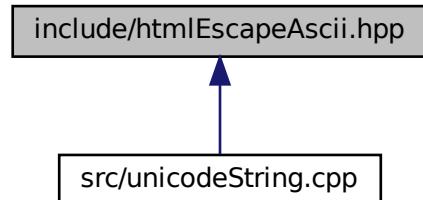
6.45 include/htmlEscapeAscii.hpp File Reference

Declare the [Tang::HtmlEscapeAscii](#) used to tokenize a Tang script.

```
#include <FlexLexer.h>
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()`

6.45.1 Detailed Description

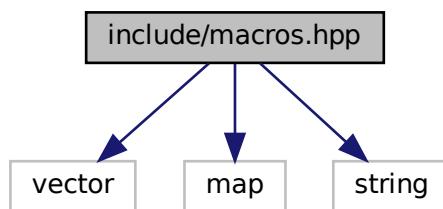
Declare the [Tang::HtmlEscapeAscii](#) used to tokenize a Tang script.

6.46 include/macros.hpp File Reference

Contains generic macros.

```
#include <vector>
#include <map>
#include <string>
```

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::NativeBoundFunction` = `GarbageCollected(*) (GarbageCollected &, std::vector< GarbageCollected > &)`
A function pointer that will be executed as bound to an object.
- using `Tang::NativeBoundFunctionMap` = `std::map< std::string, NativeBoundFunction >`
A map of method names to NativeBoundFunction objects.

6.46.1 Detailed Description

Contains generic macros.

6.47 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` ,
`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` }

6.47.1 Detailed Description

Declare the Opcodes used in the Bytecode representation of a program.

6.47.2 Enumeration Type Documentation

6.47.2.1 Opcode

```
enum Tang::Opcode [ strong ]
```

Enumerator

POP	Pop a val.
PEEK	Stack # (from fp): push val from stack #.
POKE	Stack # (from fp): Copy a val, store @ stack #.
COPY	Stack # (from fp): Deep copy val @ stack #, store @ stack #.
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.
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].

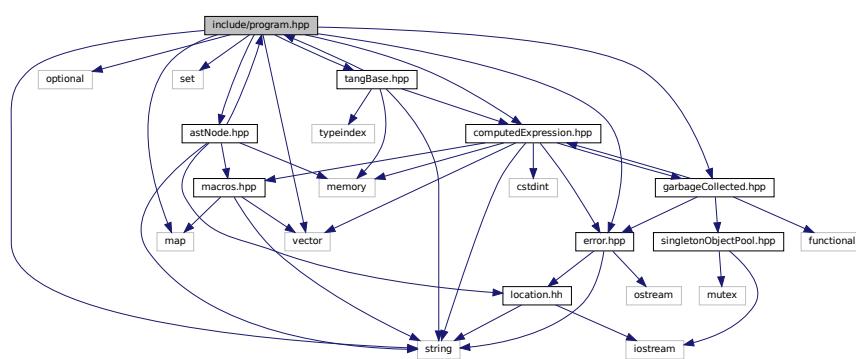
Enumerator

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.

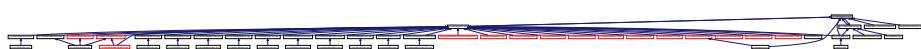
6.48 include/program.hpp File Reference

Declare the [Tang::Program](#) class used to compile and execute source code.

```
#include <string>
#include <optional>
#include <vector>
#include <set>
#include <map>
#include "astNode.hpp"
#include "error.hpp"
#include "tangBase.hpp"
#include "computedExpression.hpp"
#include "garbageCollected.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.

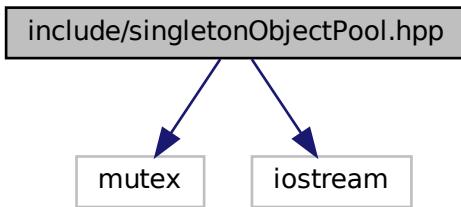
6.48.1 Detailed Description

Declare the [Tang::Program](#) class used to compile and execute source code.

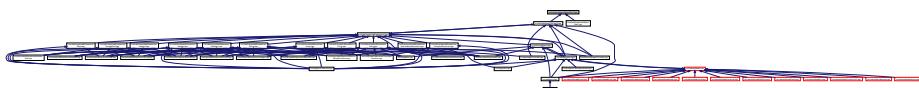
6.49 include/singletonObjectPool.hpp File Reference

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.

6.49.1 Detailed Description

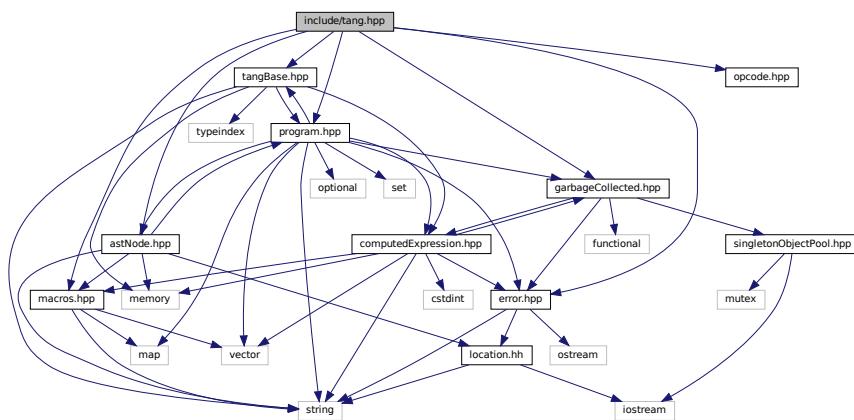
Declare the `Tang::SingletonObjectPool` class.

6.50 include/tang.hpp File Reference

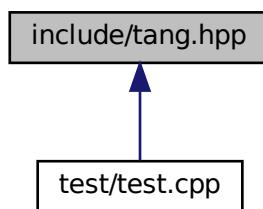
Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

```
#include "macros.hpp"
#include "tangBase.hpp"
#include "astNode.hpp"
#include "error.hpp"
#include "garbageCollected.hpp"
#include "program.hpp"
#include "opcode.hpp"
```

Include dependency graph for tang.hpp:



This graph shows which files directly or indirectly include this file:



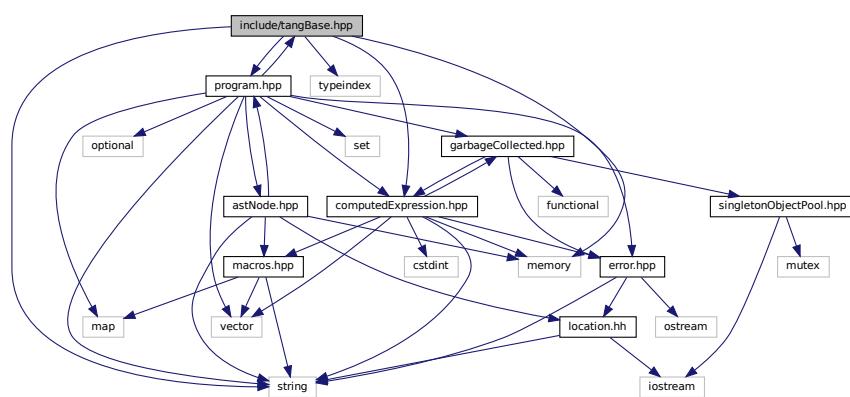
6.50.1 Detailed Description

Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

6.51 include/tangBase.hpp File Reference

Declare the [Tang::TangBase](#) class used to interact with Tang.

```
#include <memory>
#include <string>
#include <typeindex>
#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.

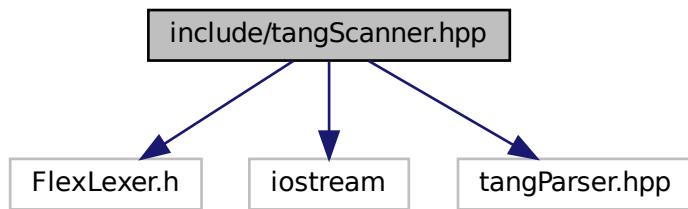
6.51.1 Detailed Description

Declare the [Tang::TangBase](#) class used to interact with Tang.

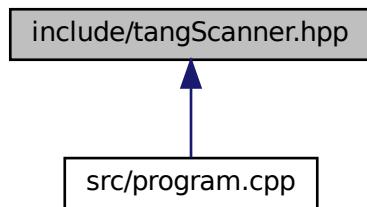
6.52 include/tangScanner.hpp File Reference

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()`

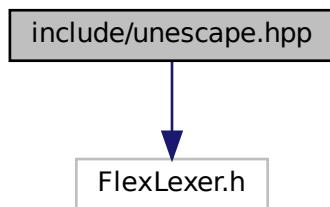
6.52.1 Detailed Description

Declare the [Tang::TangScanner](#) used to tokenize a Tang script.

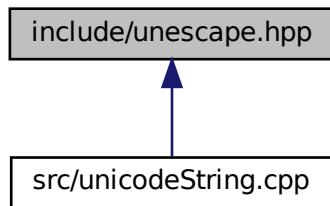
6.53 include/unescape.hpp File Reference

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()`

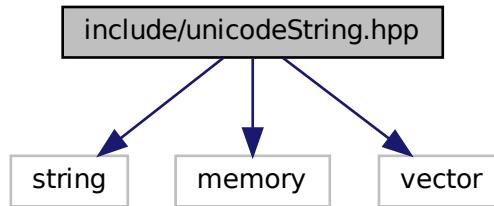
6.53.1 Detailed Description

Declare the [Tang::Unescape](#) used to tokenize a Tang script.

6.54 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)
Return an Ascii-only, "html escaped" version of the provided string.

6.54.1 Detailed Description

Contains the code to interface with the ICU library.

6.54.2 Function Documentation

6.54.2.1 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

<i>str</i>	The string to be escaped.
------------	---------------------------

Returns

An "escaped" version of the provided string.

Here is the call graph for this function:



6.54.2.2 htmlEscapeAscii()

```
string Tang::htmlEscapeAscii (
    const std::string & str )
```

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.

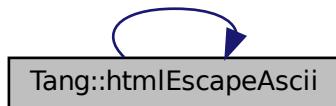
Parameters

<code>str</code>	The string to be escaped.
------------------	---------------------------

Returns

An "escaped" version of the provided string.

Here is the call graph for this function:



6.54.2.3 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.

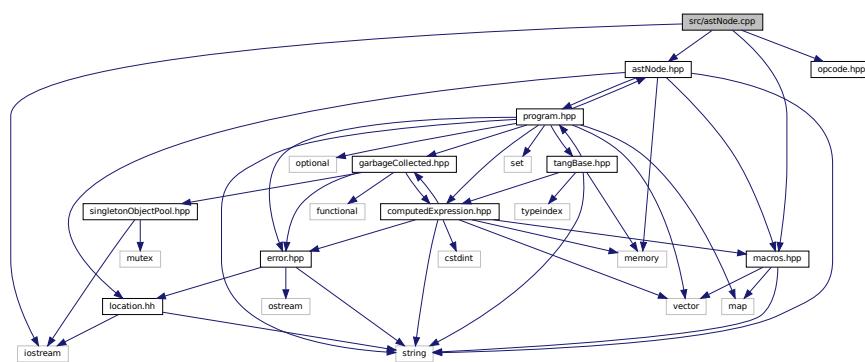
Here is the call graph for this function:



6.55 src/astNode.cpp File Reference

Define the [Tang::AstNode](#) class.

```
#include <iostream>
#include "macros.hpp"
#include "astNode.hpp"
#include "opcode.hpp"
Include dependency graph for astNode.cpp:
```



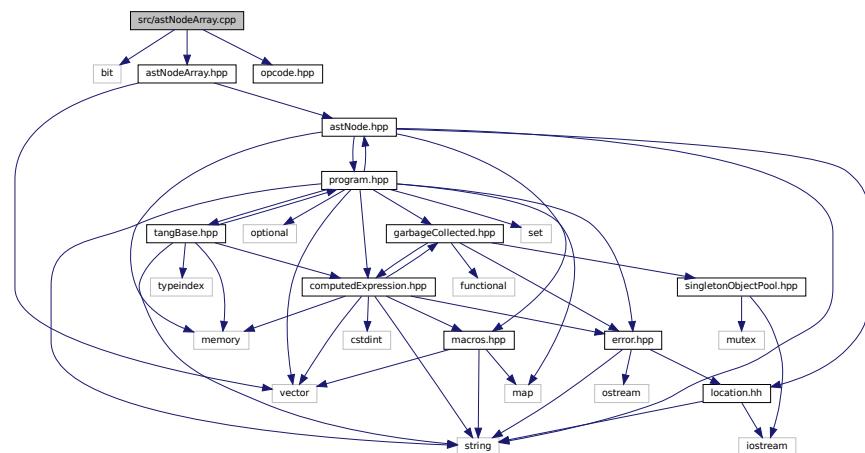
6.55.1 Detailed Description

Define the [Tang::AstNode](#) class.

6.56 src/astNodeArray.cpp File Reference

Define the [Tang::AstNodeArray](#) class.

```
#include <bit>
#include "astNodeArray.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeArray.cpp:
```



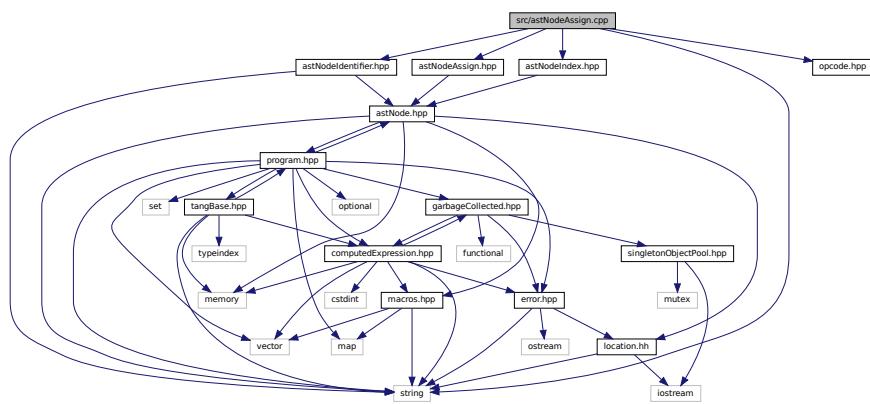
6.56.1 Detailed Description

Define the [Tang::AstNodeArray](#) class.

6.57 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 dependency graph for astNodeAssign.cpp:
```



6.57.1 Detailed Description

Define the [Tang::AstNodeAssign](#) class.

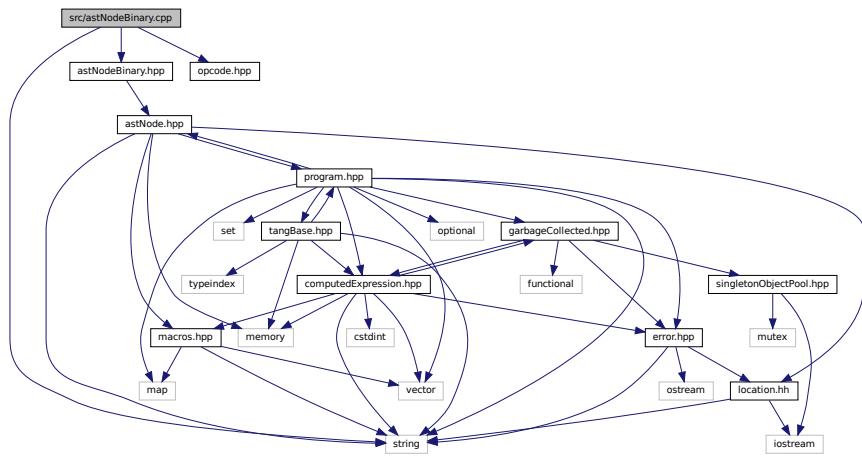
6.58 src/astNodeBinary.cpp File Reference

Define the [Tang::AstNodeBinary](#) class.

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

```
#include "opcode.hpp"
```

Include dependency graph for astNodeBinary.cpp:



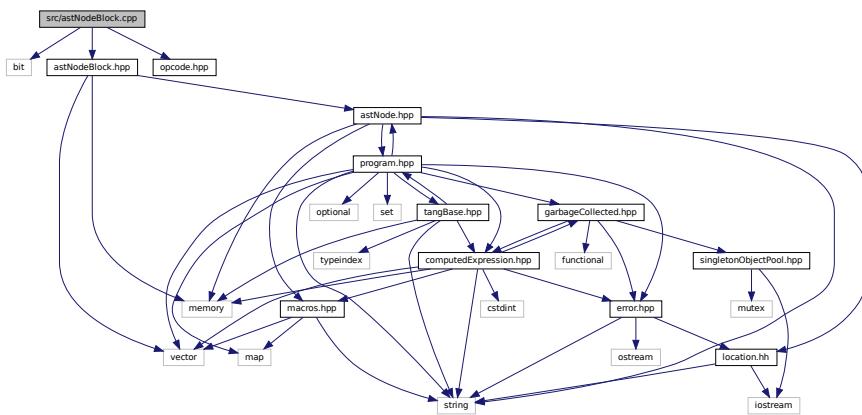
6.58.1 Detailed Description

Define the `Tang::AstNodeBinary` class.

6.59 src/astNodeBlock.cpp File Reference

Define the `Tang::AstNodeBlock` class.

```
#include <bit>
#include "astNodeBlock.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeBlock.cpp:
```



6.59.1 Detailed Description

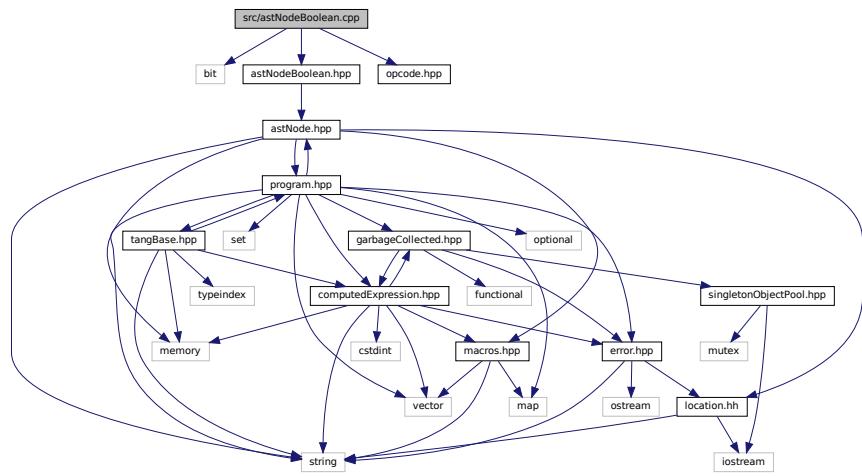
Define the `Tang::AstNodeBlock` class.

6.60 src/astNodeBoolean.cpp File Reference

Define the [Tang::AstNodeBoolean](#) class.

```
#include <bit>
#include "astNodeBoolean.hpp"
#include "opcode.hpp"
```

Include dependency graph for astNodeBoolean.cpp:



6.60.1 Detailed Description

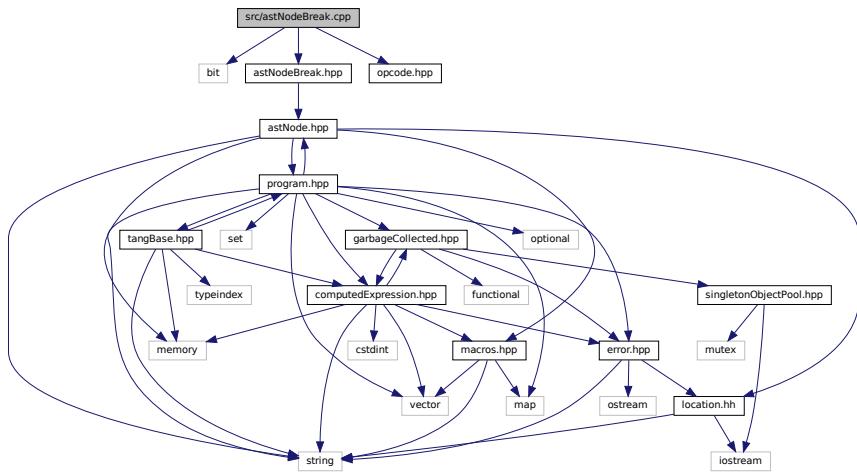
Define the [Tang::AstNodeBoolean](#) class.

6.61 src/astNodeBreak.cpp File Reference

Define the [Tang::AstNodeBreak](#) class.

```
#include <bit>
#include "astNodeBreak.hpp"
#include "opcode.hpp"
```

Include dependency graph for `astNodeBreak.cpp`:



6.61.1 Detailed Description

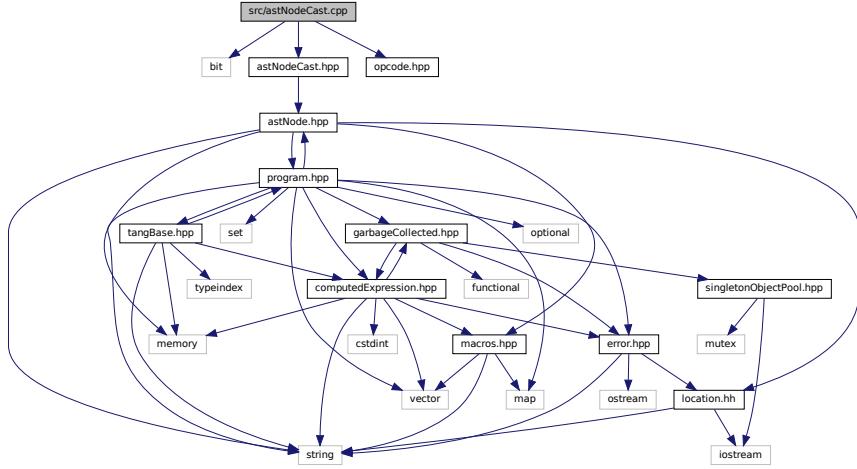
Define the [Tang::AstNodeBreak](#) class.

6.62 src/astNodeCast.cpp File Reference

Define the [Tang::AstNodeCast](#) class.

```
#include <bit>
#include "astNodeCast.hpp"
#include "opcode.hpp"
```

Include dependency graph for `astNodeCast.cpp`:



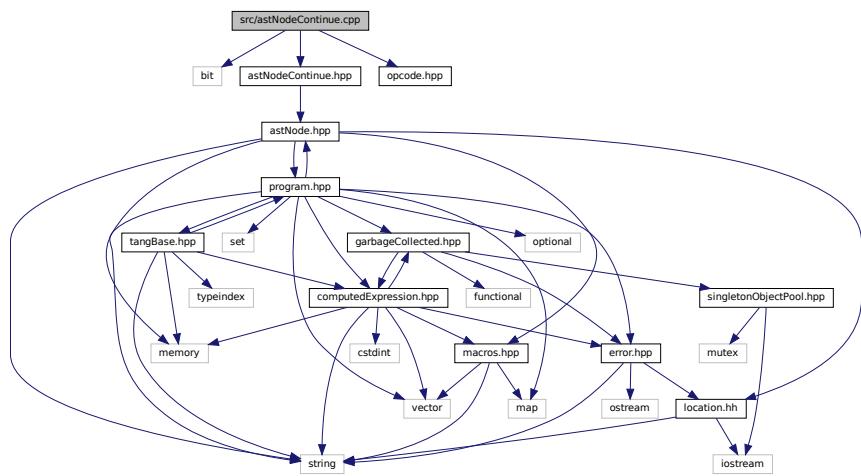
6.62.1 Detailed Description

Define the [Tang::AstNodeCast](#) class.

6.63 src/astNodeContinue.cpp File Reference

Define the [Tang::AstNodeContinue](#) class.

```
#include <bit>
#include "astNodeContinue.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeContinue.cpp:
```



6.63.1 Detailed Description

Define the [Tang::AstNodeContinue](#) class.

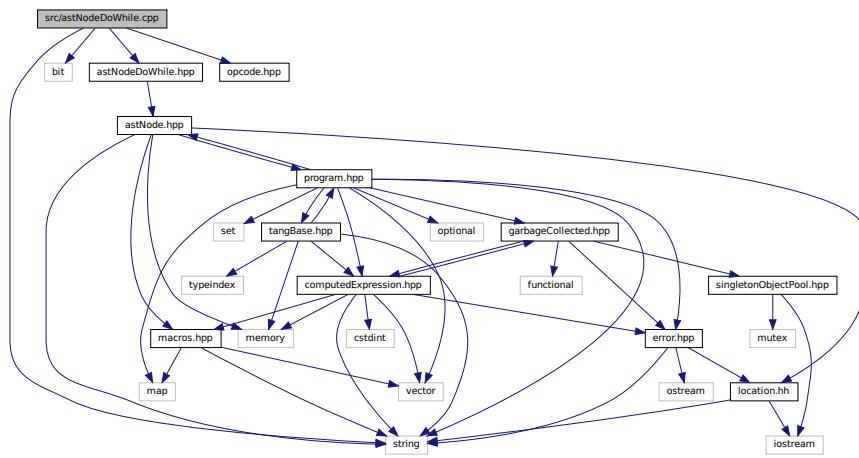
6.64 src/astNodeDoWhile.cpp File Reference

Define the [Tang::AstNodeDoWhile](#) class.

```
#include <string>
#include <bit>
#include "astNodeDoWhile.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for `astNodeDoWhile.cpp`:



6.64.1 Detailed Description

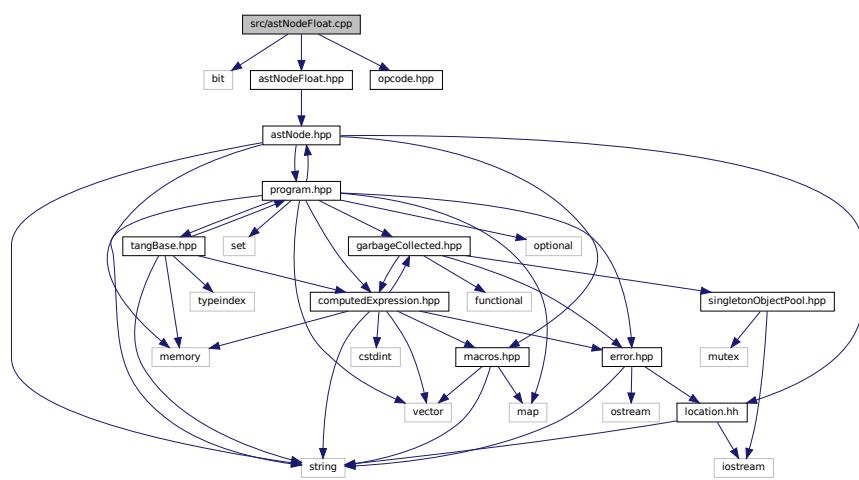
Define the [Tang::AstNodeDoWhile](#) class.

6.65 src/astNodeFloat.cpp File Reference

Define the [Tang::AstNodeFloat](#) class.

```
#include <bit>
#include "astNodeFloat.hpp"
#include "opcode.hpp"
```

Include dependency graph for `astNodeFloat.cpp`:



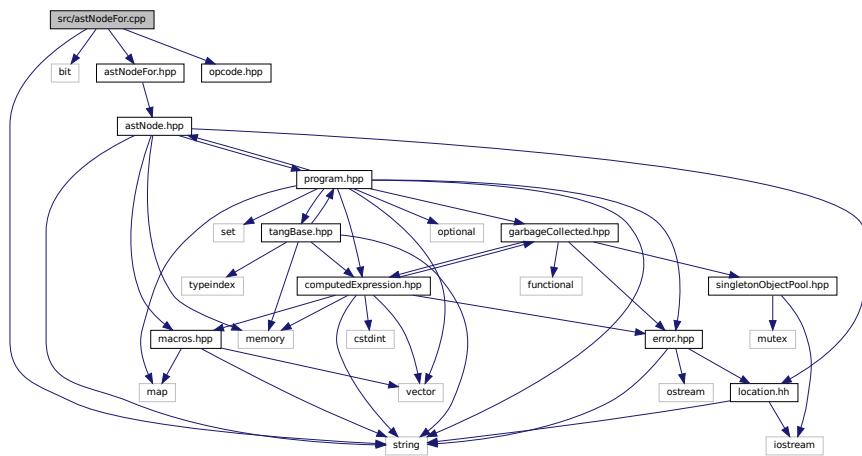
6.65.1 Detailed Description

Define the [Tang::AstNodeFloat](#) class.

6.66 src/astNodeFor.cpp File Reference

Define the [Tang::AstNodeFor](#) class.

```
#include <string>
#include <bit>
#include "astNodeFor.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeFor.cpp:
```



6.66.1 Detailed Description

Define the [Tang::AstNodeFor](#) class.

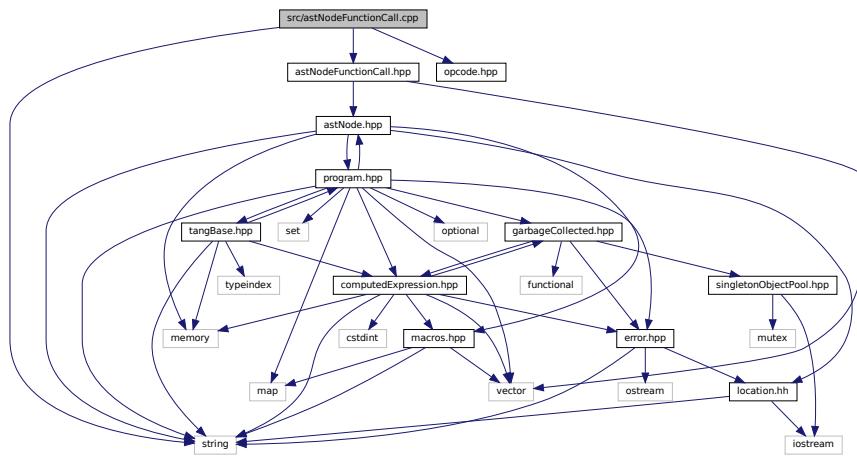
6.67 src/astNodeFunctionCall.cpp File Reference

Define the [Tang::AstNodeFunctionCall](#) class.

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

```
#include "opcode.hpp"
```

Include dependency graph for `astNodeFunctionCall.cpp`:



6.67.1 Detailed Description

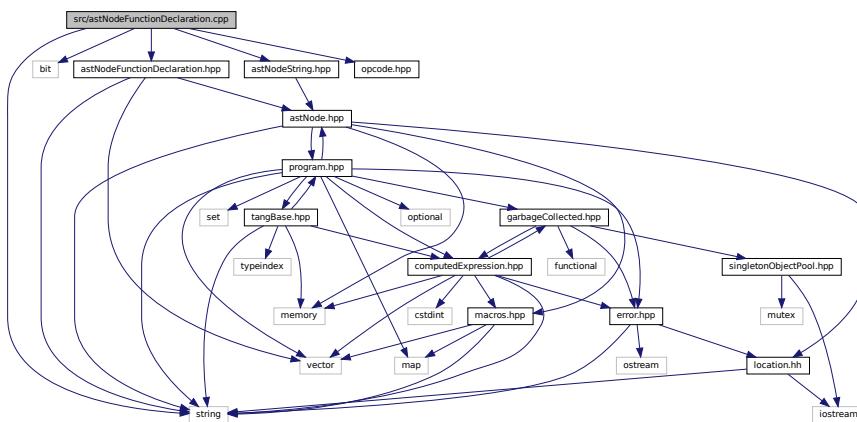
Define the [Tang::AstNodeFunctionCall](#) class.

6.68 src/astNodeFunctionDeclaration.cpp File Reference

Define the [Tang::AstNodeFunctionDeclaration](#) class.

```
#include <string>
#include <bit>
#include "astNodeFunctionDeclaration.hpp"
#include "astNodeString.hpp"
#include "opcode.hpp"
```

Include dependency graph for `astNodeFunctionDeclaration.cpp`:



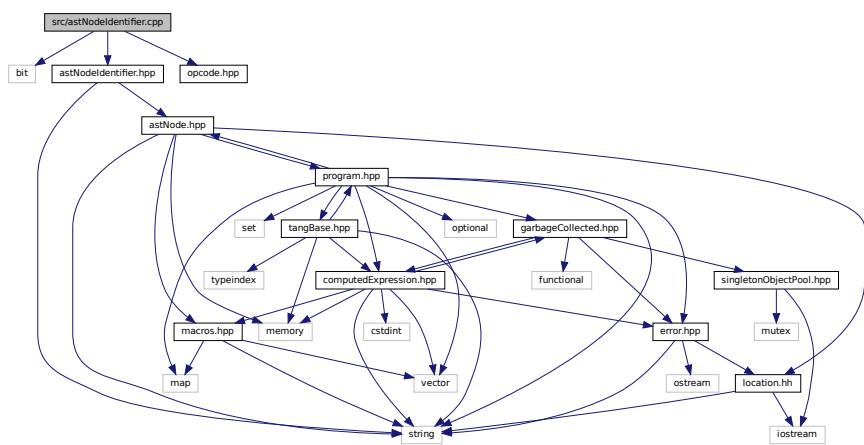
6.68.1 Detailed Description

Define the [Tang::AstNodeFunctionDeclaration](#) class.

6.69 src/astNodelIdentifier.cpp File Reference

Define the [Tang::AstNodelIdentifier](#) class.

```
#include <bit>
#include "astNodelIdentifier.hpp"
#include "opcode.hpp"
Include dependency graph for astNodelIdentifier.cpp:
```



6.69.1 Detailed Description

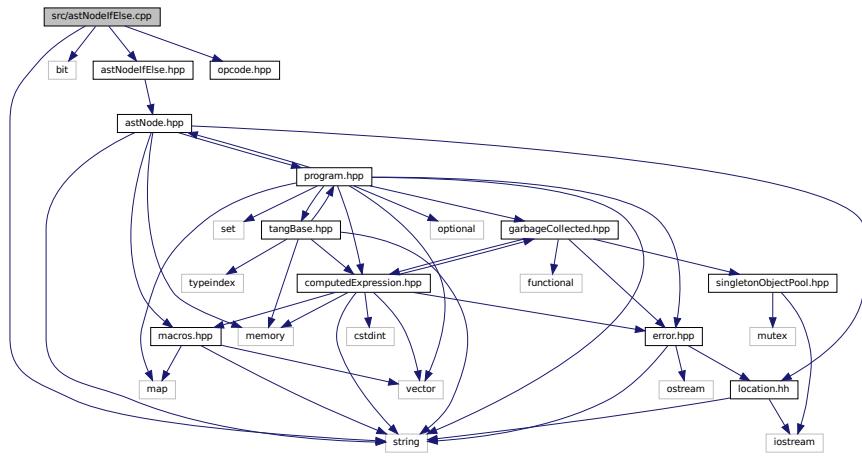
Define the [Tang::AstNodelIdentifier](#) class.

6.70 src/astNodelIfElse.cpp File Reference

Define the [Tang::AstNodelIfElse](#) class.

```
#include <string>
#include <bit>
#include "astNodelIfElse.hpp"
```

```
#include "opcode.hpp"
Include dependency graph for astNodelfElse.cpp:
```



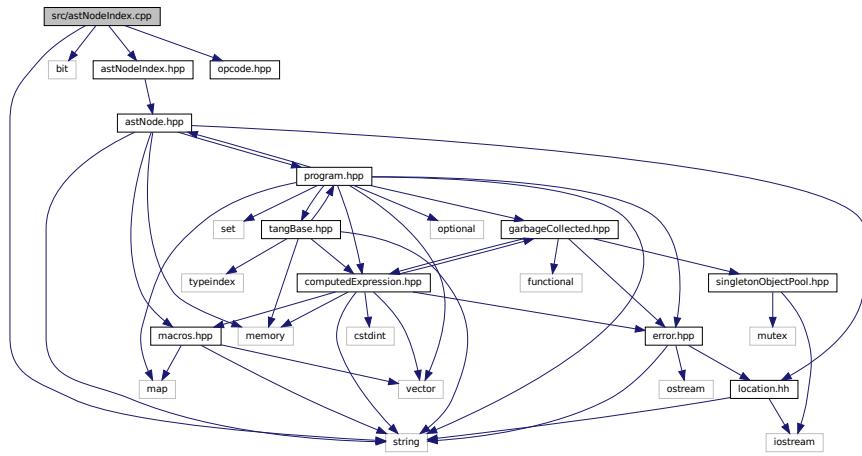
6.70.1 Detailed Description

Define the `Tang::AstNodeIfElse` class.

6.71 src/astNodeIndex.cpp File Reference

Define the `Tang::AstNodeIndex` class.

```
#include <string>
#include <bit>
#include "astNodeIndex.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeIndex.hpp:
```



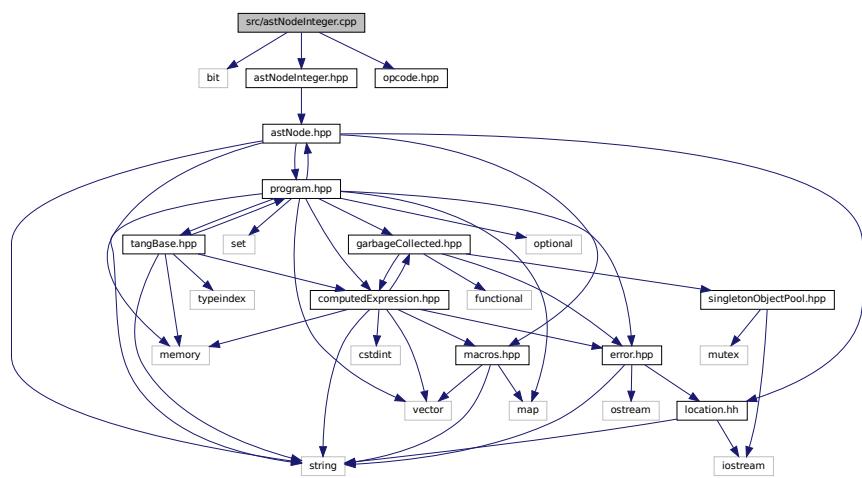
6.71.1 Detailed Description

Define the [Tang::AstNodeIndex](#) class.

6.72 src/astNodeInteger.cpp File Reference

Define the [Tang::AstNodeInteger](#) class.

```
#include <bit>
#include "astNodeInteger.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeInteger.cpp:
```



6.72.1 Detailed Description

Define the [Tang::AstNodeInteger](#) class.

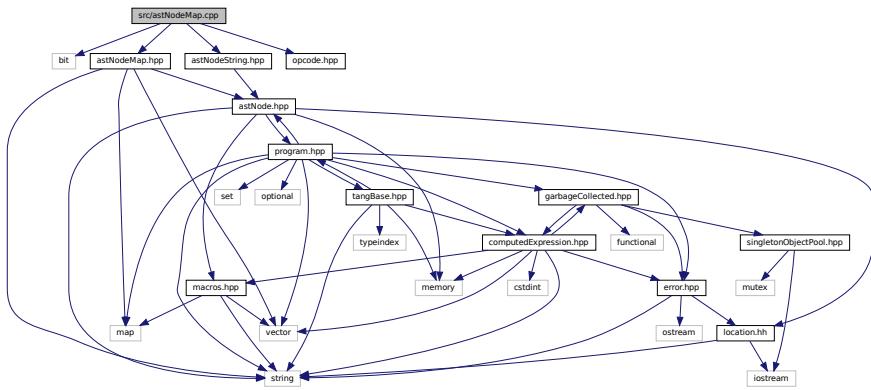
6.73 src/astNodeMap.cpp File Reference

Define the [Tang::AstNodeMap](#) class.

```
#include <bit>
#include "astNodeMap.hpp"
#include "astNodeString.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeMap.cpp:



6.73.1 Detailed Description

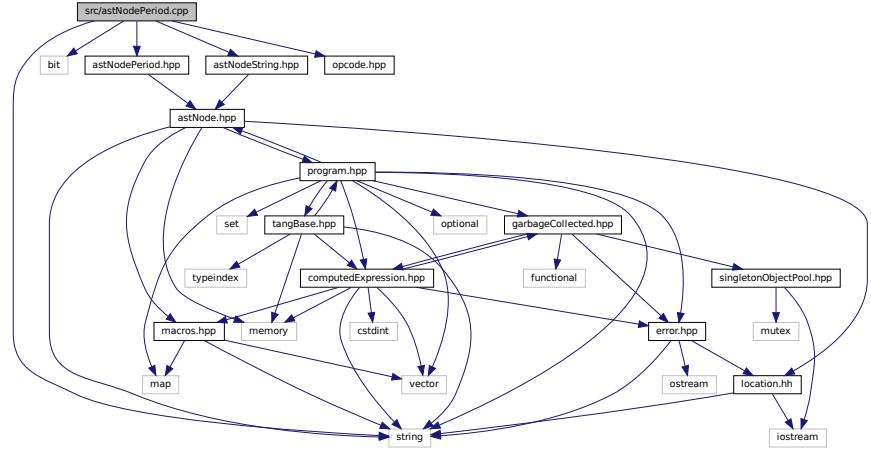
Define the `Tang::AstNodeMap` class.

6.74 src/astNodePeriod.cpp File Reference

Define the `Tang::AstNodePeriod` class.

```
#include <string>
#include <bit>
#include "astNodePeriod.hpp"
#include "astNodeString.hpp"
#include "opcode.hpp"
Include dependency graph for astNodePeriod.cpp:
```

Include dependency graph for astNodePeriod.cpp:



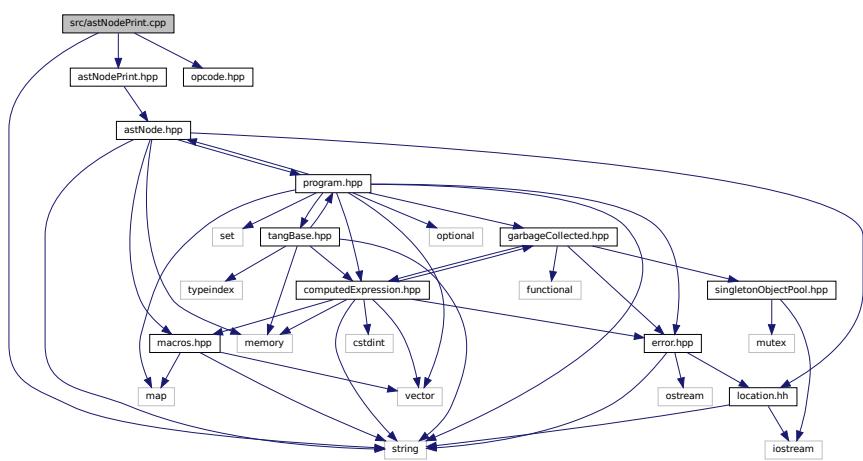
6.74.1 Detailed Description

Define the [Tang::AstNodePeriod](#) class.

6.75 src/astNodePrint.cpp File Reference

Define the [Tang::AstNodePrint](#) class.

```
#include <string>
#include "astNodePrint.hpp"
#include "opcode.hpp"
Include dependency graph for astNodePrint.cpp:
```



6.75.1 Detailed Description

Define the [Tang::AstNodePrint](#) class.

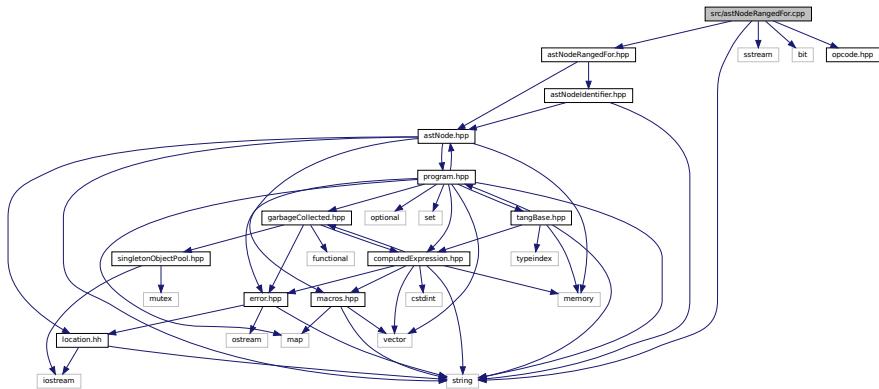
6.76 src/astNodeRangedFor.cpp File Reference

Define the [Tang::AstNodeRangedFor](#) class.

```
#include <string>
#include <sstream>
#include <bit>
#include "astNodeRangedFor.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeRangedFor.cpp:



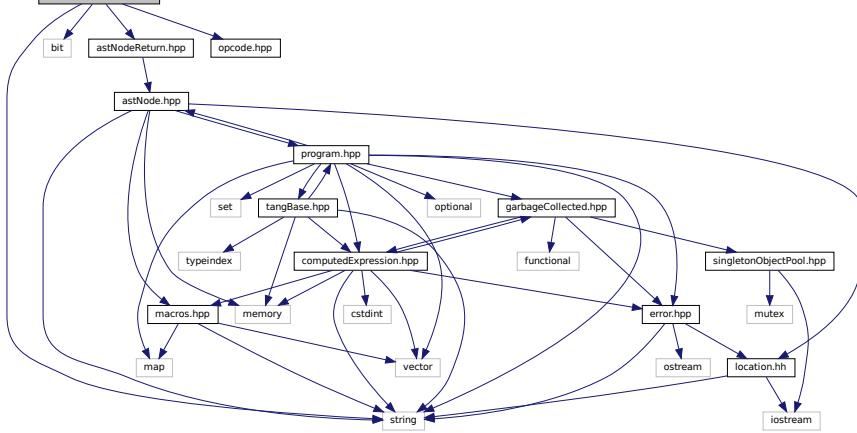
6.76.1 Detailed Description

Define the `Tang::AstNodeRangedFor` class.

6.77 src/astNodeReturn.cpp File Reference

Define the `Tang::AstNodeReturn` class.

```
#include <string>
#include <bit>
#include "astNodeReturn.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeReturn.cpp:
```



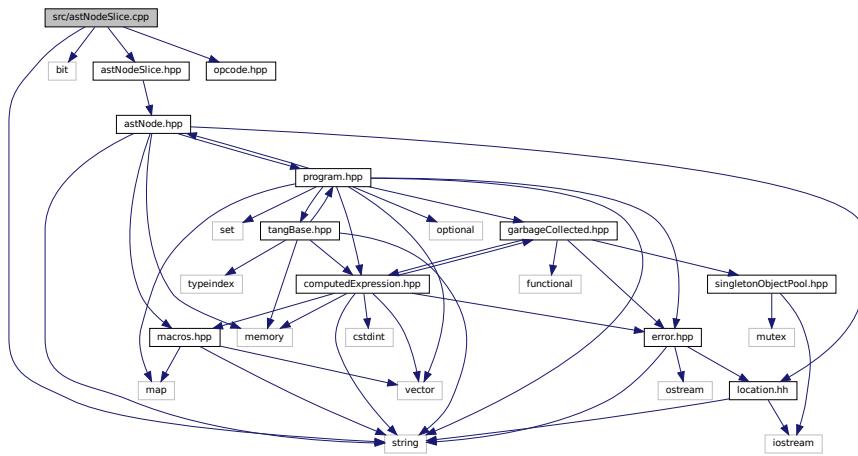
6.77.1 Detailed Description

Define the `Tang::AstNodeReturn` class.

6.78 src/astNodeSlice.cpp File Reference

Define the [Tang::AstNodeSlice](#) class.

```
#include <string>
#include <bit>
#include "astNodeSlice.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeSlice.cpp:
```



6.78.1 Detailed Description

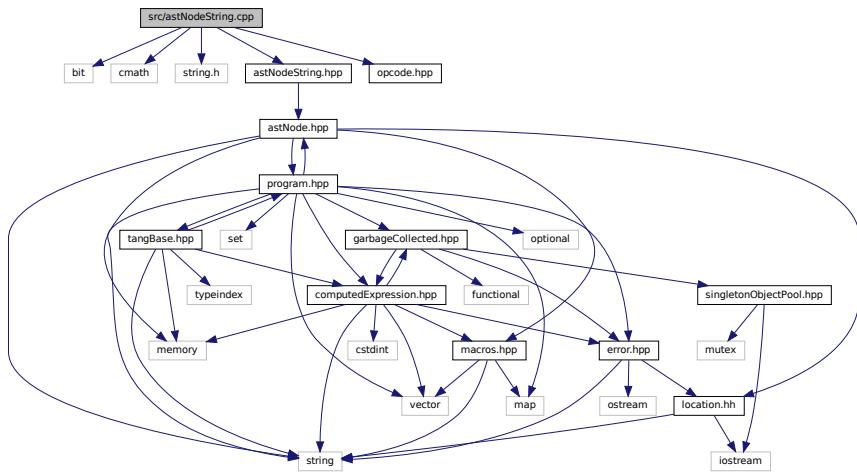
Define the [Tang::AstNodeSlice](#) class.

6.79 src/astNodeString.cpp File Reference

Define the [Tang::AstNodeString](#) class.

```
#include <bit>
#include <cmath>
#include <string.h>
#include "astNodeString.hpp"
#include "opcode.hpp"
```

Include dependency graph for `astNodeString.cpp`:



6.79.1 Detailed Description

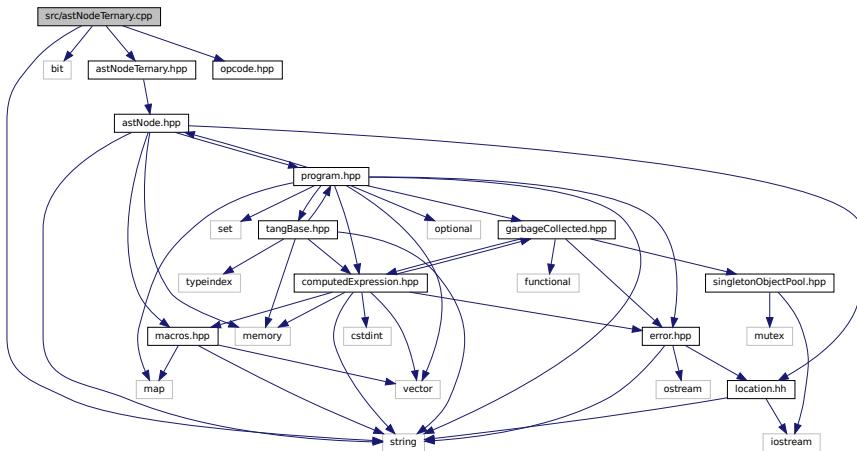
Define the [Tang::AstNodeString](#) class.

6.80 src/astNodeTernary.cpp File Reference

Define the [Tang::AstNodeTernary](#) class.

```
#include <string>
#include <bit>
#include "astNodeTernary.hpp"
#include "opcode.hpp"
```

Include dependency graph for `astNodeTernary.cpp`:



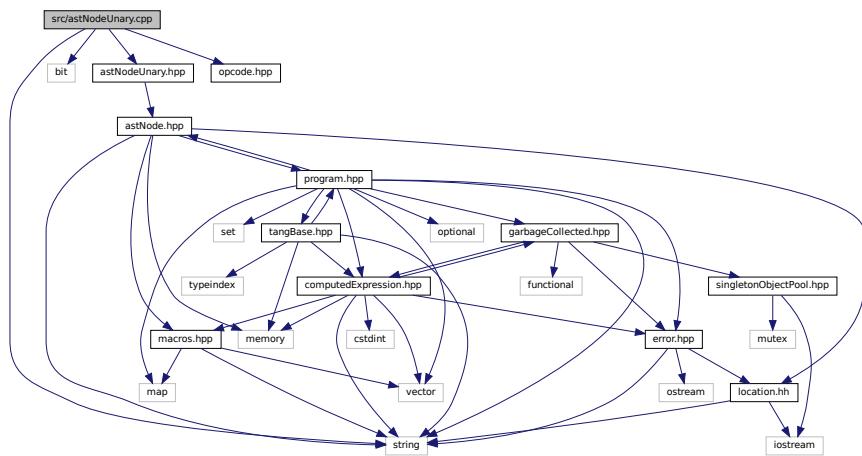
6.80.1 Detailed Description

Define the [Tang::AstNodeTernary](#) class.

6.81 src/astNodeUnary.cpp File Reference

Define the [Tang::AstNodeUnary](#) class.

```
#include <string>
#include <bit>
#include "astNodeUnary.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeUnary.cpp:
```



6.81.1 Detailed Description

Define the [Tang::AstNodeUnary](#) class.

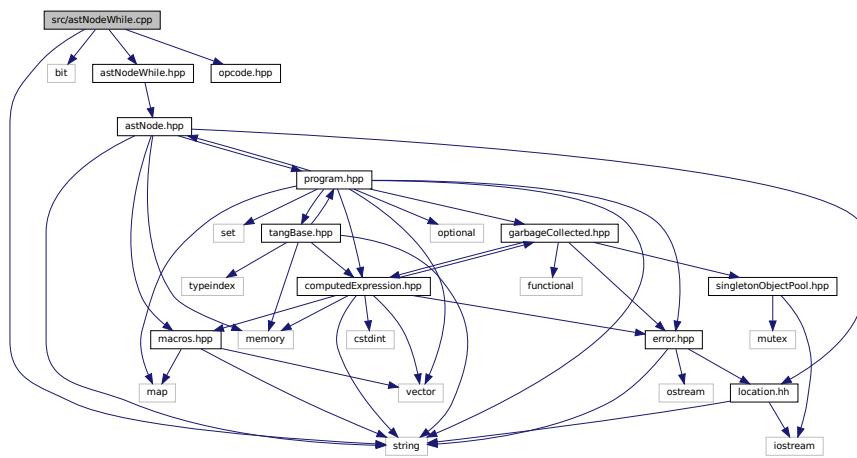
6.82 src/astNodeWhile.cpp File Reference

Define the [Tang::AstNodeWhile](#) class.

```
#include <string>
#include <bit>
#include "astNodeWhile.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for `astNodeWhile.cpp`:



6.82.1 Detailed Description

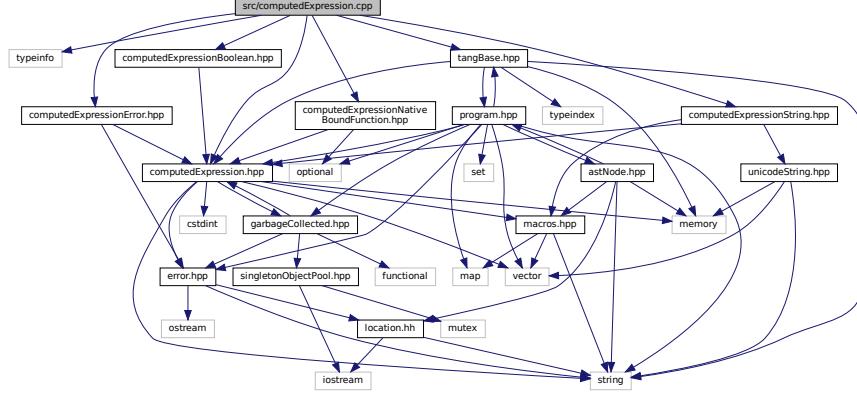
Define the [Tang::AstNodeWhile](#) class.

6.83 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`:



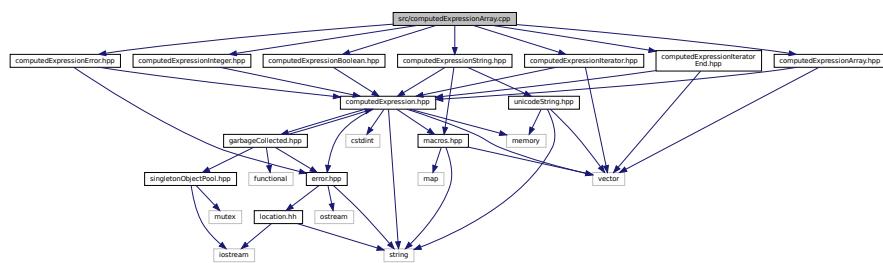
6.83.1 Detailed Description

Define the [Tang::ComputedExpression](#) class.

6.84 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:
```



6.84.1 Detailed Description

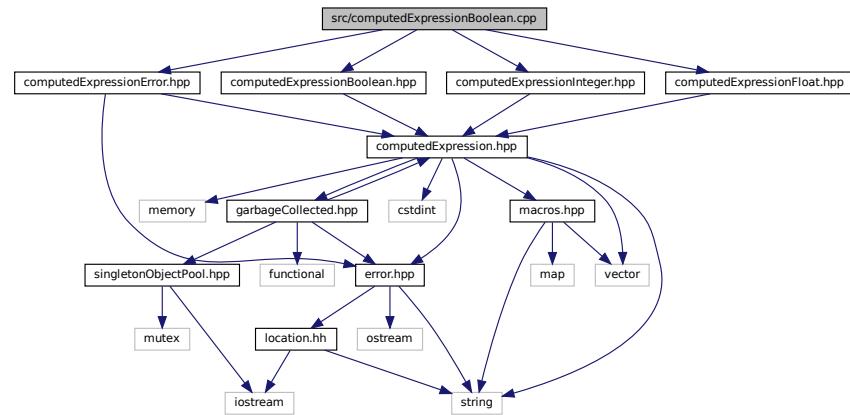
Define the [Tang::ComputedExpressionArray](#) class.

6.85 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:
```



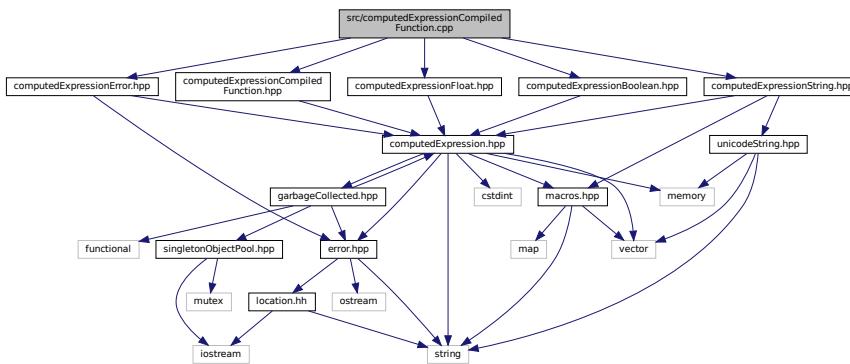
6.85.1 Detailed Description

Define the [Tang::ComputedExpressionBoolean](#) class.

6.86 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:
```



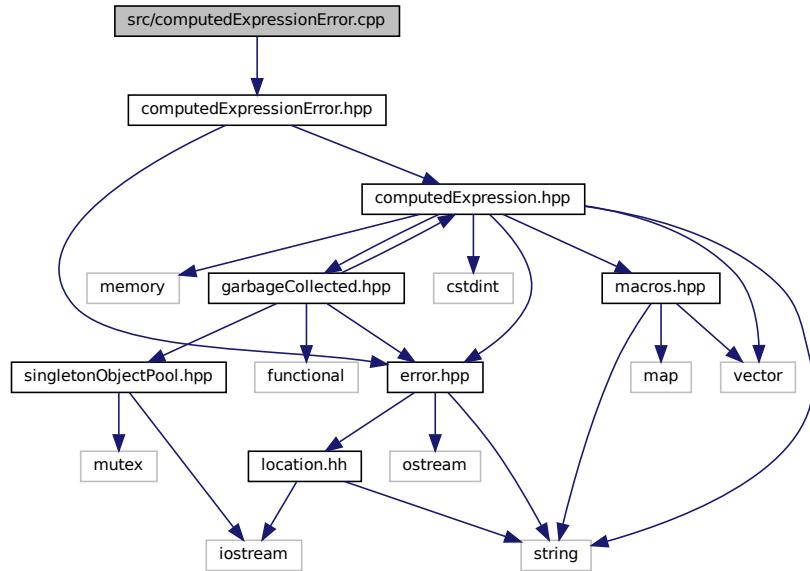
6.86.1 Detailed Description

Define the [Tang::ComputedExpressionCompiledFunction](#) class.

6.87 src/computedExpressionError.cpp File Reference

Define the `Tang::ComputedExpressionError` class.

```
#include "computedExpressionError.hpp"
Include dependency graph for computedExpressionError.cpp:
```



6.87.1 Detailed Description

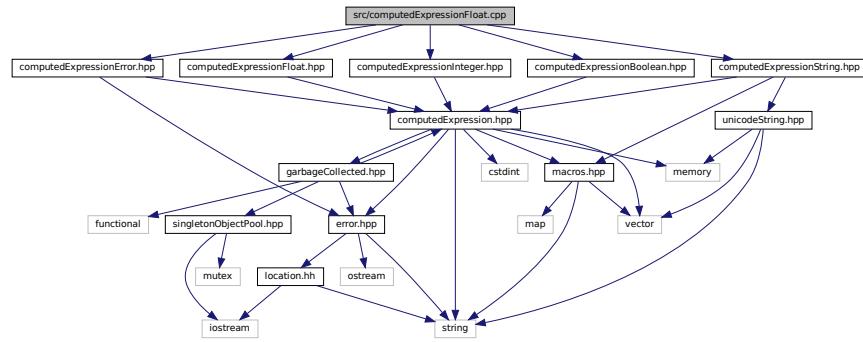
Define the `Tang::ComputedExpressionError` class.

6.88 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:
```



6.88.1 Detailed Description

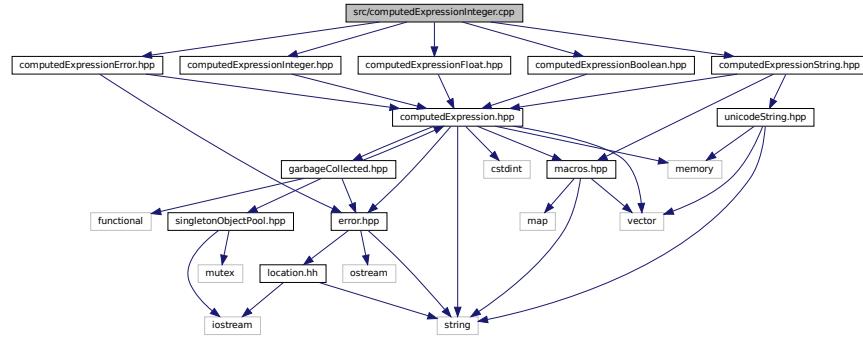
Define the `Tang::ComputedExpressionFloat` class.

6.89 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.hpp:
```

Include dependency graph for computedExpressionInteger.cpp:



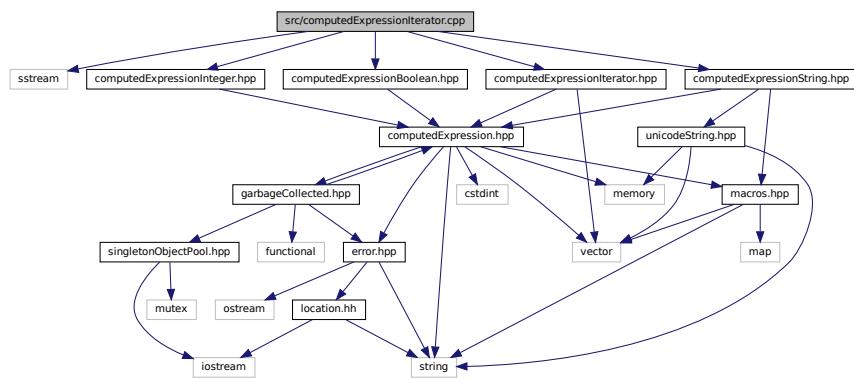
6.89.1 Detailed Description

Define the `Tang::ComputedExpressionInteger` class.

6.90 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:
```



6.90.1 Detailed Description

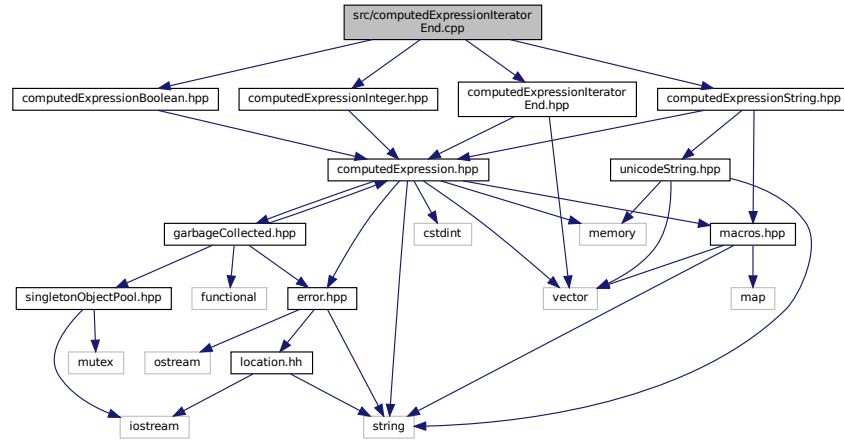
Define the [Tang::ComputedExpressionIterator](#) class.

6.91 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:



6.91.1 Detailed Description

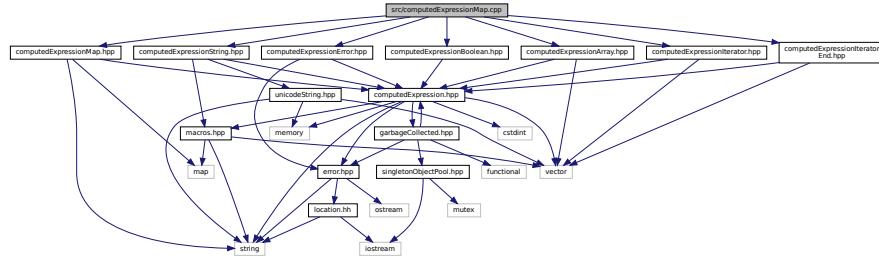
Define the `Tang::ComputedExpressionIteratorEnd` class.

6.92 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:



6.92.1 Detailed Description

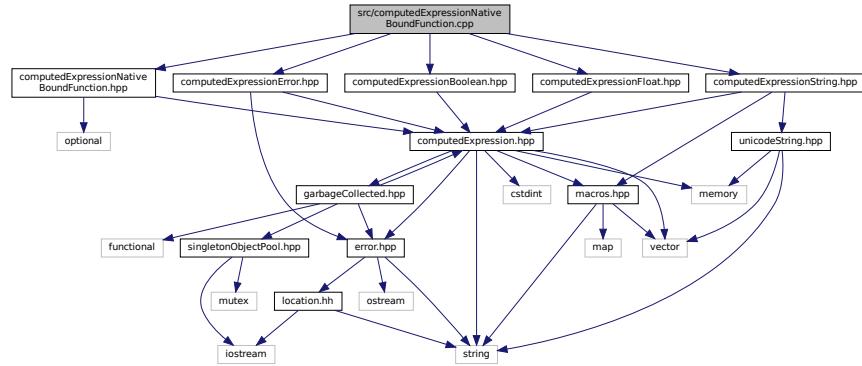
Define the `Tang::ComputedExpressionMap` class.

6.93 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`:



6.93.1 Detailed Description

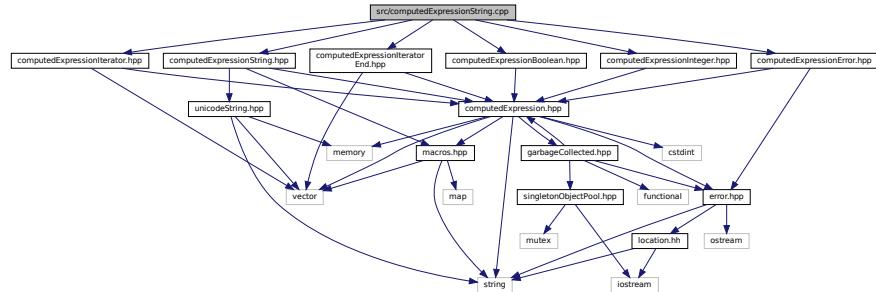
Define the [Tang::ComputedExpressionNativeBoundFunction](#) class.

6.94 src/computedExpressionString.cpp File Reference

Define the [Tang::ComputedExpressionString](#) class.

```
#include "computedExpressionString.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionError.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionIterator.hpp"
#include "computedExpressionIteratorEnd.hpp"
```

Include dependency graph for `computedExpressionString.cpp`:



6.94.1 Detailed Description

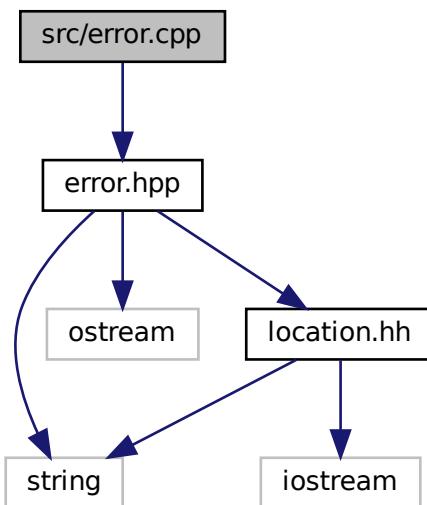
Define the [Tang::ComputedExpressionString](#) class.

6.95 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)`

6.95.1 Detailed Description

Define the [Tang::Error](#) class.

6.95.2 Function Documentation

6.95.2.1 operator<<()

```
std::ostream& Tang::operator<< (
    std::ostream & out,
    const Error & error )
```

Parameters

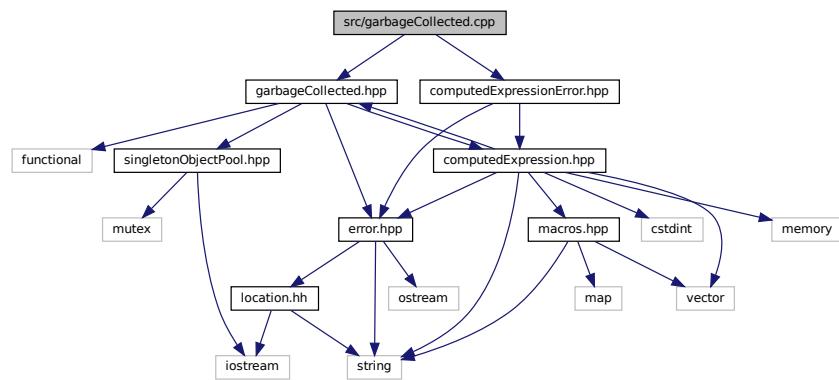
<i>out</i>	The output stream.
<i>error</i>	The Error object.

Returns

The output stream.

6.96 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)

6.96.1 Function Documentation

6.96.1.1 operator<<()

```
std::ostream& Tang::operator<< (
    std::ostream & out,
    const GarbageCollected & gc )
```

Parameters

<i>out</i>	The output stream.
<i>gc</i>	The GarbageCollected value.

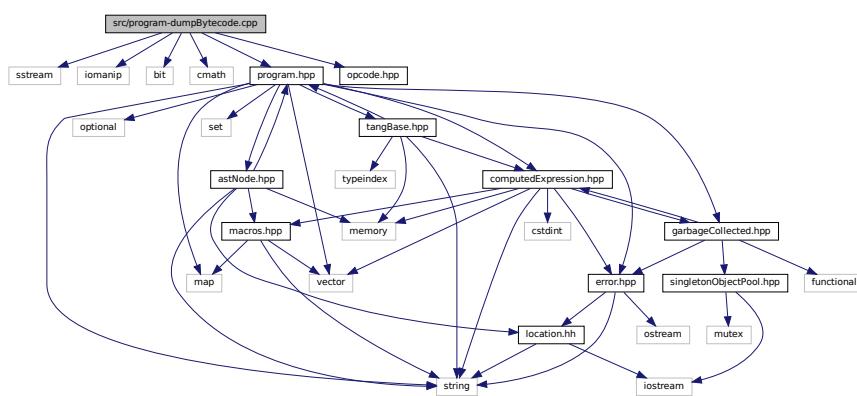
Returns

The output stream.

6.97 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.

6.97.1 Detailed Description

Define the `Tang::Program::dumpBytecode` method.

6.97.2 Macro Definition Documentation

6.97.2.1 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

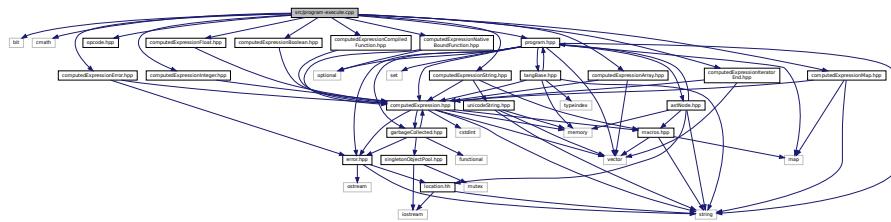
x	The number of additional vector entries that should exist.
---	--

6.98 src/program-execute.cpp File Reference

Define the [Tang::Program::execute](#) method.

```
#include <bit>
#include <cmath>
#include "program.hpp"
#include "opcode.hpp"
#include "computedExpressionError.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionArray.hpp"
#include "computedExpressionMap.hpp"
#include "computedExpressionCompiledFunction.hpp"
#include "computedExpressionNativeBoundFunction.hpp"
#include "computedExpressionIteratorEnd.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.

6.98.1 Detailed Description

Define the [Tang::Program::execute](#) method.

6.98.2 Macro Definition Documentation

6.98.2.1 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.
---	--

6.98.2.2 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

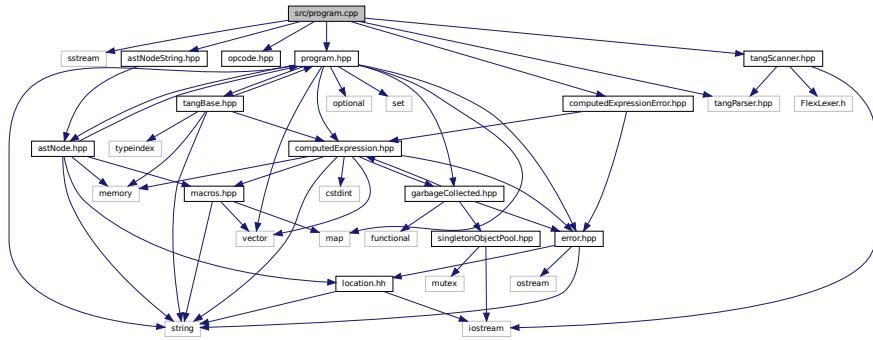
x	The number of entries that should exist in the stack.
---	---

6.99 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 "computedExpressionError.hpp"
```

Include dependency graph for program.cpp:



6.99.1 Detailed Description

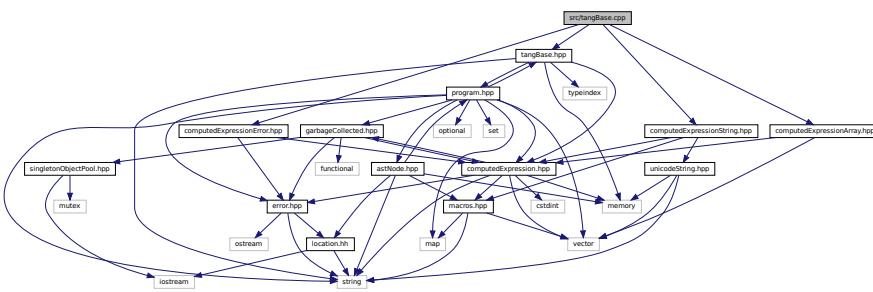
Define the [Tang::Program](#) class.

6.100 src/tangBase.cpp File Reference

Define the [Tang::TangBase](#) class.

```
#include "tangBase.hpp"
#include "computedExpressionArray.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionError.hpp"
```

Include dependency graph for tangBase.cpp:



6.100.1 Detailed Description

Define the [Tang::TangBase](#) class.

6.101 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:
```

```

graph TD
    unicodeString["src/unicodeString.cpp"] --> cassert
    unicodeString --> unicodeStringH["unicodeString.hpp"]
    unicodeString --> algorithm
    unicodeString --> sstream
    unicodeString --> unicodeUconfigH["unicode/uconfig.h"]
    unicodeString --> unicodeUStringH["unicode/ustring.h"]
    unicodeString --> unicodeBrkiterH["unicode/brkiter.h"]
    unicodeString --> unescapeH["unescape.hpp"]
    unicodeString --> htmlEscapeH["htmlEscape.hpp"]
    unicodeString --> htmlEscapeAsciiH["htmlEscapeAscii.hpp"]

    cassert --> vector
    cassert --> string
    cassert --> memory

    vector --> string
    vector --> memory

    string --> memory

    FlexLexerH["FlexLexer.h"] --> htmlEscapeH

```

The diagram shows the dependencies of the file `src/unicodeString.cpp`. It includes dependencies on `cassert`, `unicodeString.hpp`, `algorithm`, `sstream`, `unicode/uconfig.h`, `unicode/ustring.h`, `unicode/brkiter.h`, `unescape.hpp`, `htmlEscape.hpp`, and `htmlEscapeAscii.hpp`. `cassert` depends on `vector`, `string`, and `memory`. `FlexLexer.h` depends on `htmlEscape.hpp`.

6.101.1 Detailed Description

Contains the function declarations for the `Tang::UnicodeString` class and the interface to ICU.

6.102 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:
```

The diagram illustrates the dependency graph for the tang project. Nodes are represented by boxes, and directed edges by arrows, indicating dependencies. The graph is organized into several main components:

- Core Headers:** tang.hpp, tangBase.hpp, program.hpp, typeIndex.hpp, opcode.hpp.
- Utility Headers:** garbageCollected.hpp, functional.hpp, error.hpp, computedExpression.hpp, astNode.hpp.
- System Headers:** singletonObjectPool.hpp, mutex.hpp, ostream.hpp, location.hpp, string.hpp, map.hpp, vector.hpp, constraint.hpp, memory.hpp, macros.hpp.
- Standard Headers:** iostream.hpp, functional.hpp, set.hpp, optional.hpp.

Key dependencies include:

- tang.hpp** depends on **tangBase.hpp**, **program.hpp**, **typeIndex.hpp**, **functional.hpp**, **error.hpp**, **computedExpression.hpp**, **astNode.hpp**, **mutex.hpp**, **location.hpp**, **string.hpp**, **map.hpp**, **vector.hpp**, **constraint.hpp**, **memory.hpp**, and **macros.hpp**.
- functional.hpp** depends on **functional** and **error.hpp**.
- error.hpp** depends on **functional**, **location.hpp**, and **string.hpp**.
- computedExpression.hpp** depends on **functional**, **error.hpp**, **vector.hpp**, and **string.hpp**.
- astNode.hpp** depends on **functional**, **error.hpp**, **vector.hpp**, and **string.hpp**.
- mutex.hpp** depends on **functional**.
- location.hpp** depends on **functional**, **error.hpp**, and **string.hpp**.
- string.hpp** depends on **functional**, **error.hpp**, **vector.hpp**, **map.hpp**, and **constraint.hpp**.
- map.hpp** depends on **functional**.
- vector.hpp** depends on **functional**.
- constraint.hpp** depends on **functional**.
- memory.hpp** depends on **functional**.
- macros.hpp** depends on **functional**.
- iostream.hpp** depends on **functional**.

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)
- int **main** (int argc, char **argv)

Variables

- auto **tang** = TangBase::make_shared()

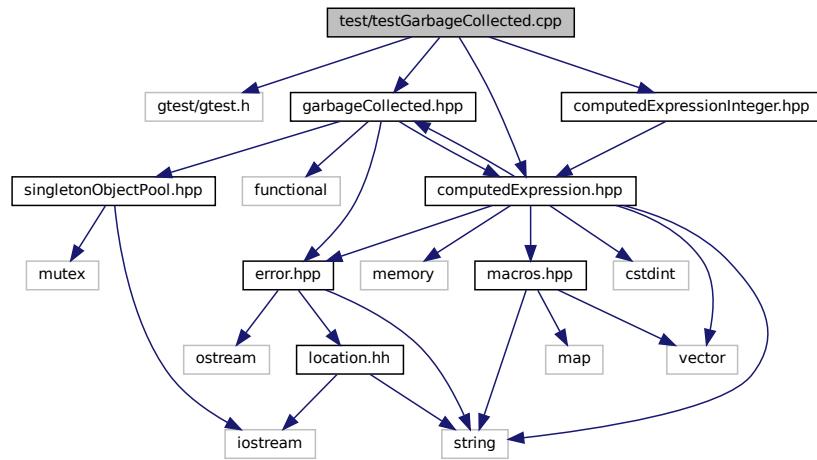
6.102.1 Detailed Description

Test the general language behaviors.

6.103 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)

6.103.1 Detailed Description

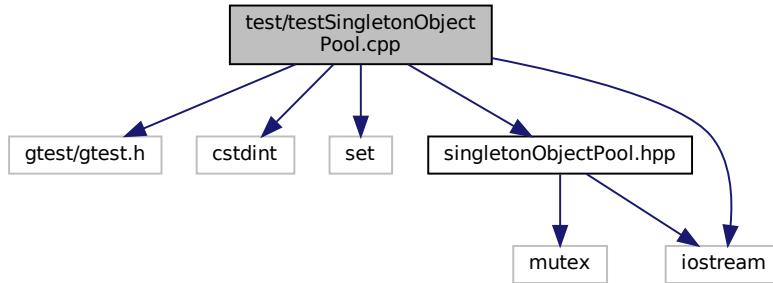
Test the generic behavior of the [Tang::GarbageCollected](#) class.

6.104 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)

6.104.1 Detailed Description

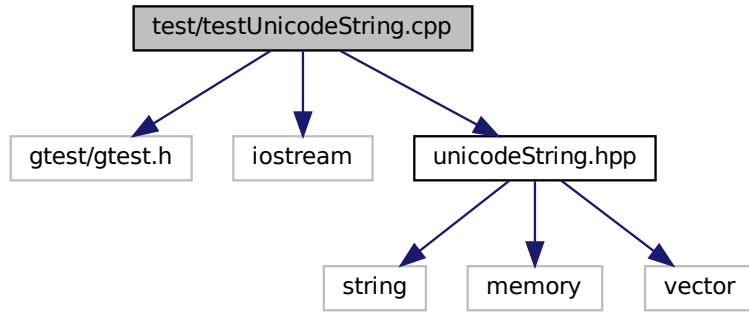
Test the generic behavior of the [Tang::SingletonObjectPool](#) class.

6.105 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)
- int `main` (int argc, char **argv)

6.105.1 Detailed Description

Contains tests for the [Tang::UnicodeString](#) class.

Index

—add
 Tang::ComputedExpression, 143
 Tang::ComputedExpressionArray, 157
 Tang::ComputedExpressionBoolean, 172
 Tang::ComputedExpressionCompiledFunction, 185
 Tang::ComputedExpressionError, 198
 Tang::ComputedExpressionFloat, 213
 Tang::ComputedExpressionInteger, 227
 Tang::ComputedExpressionIterator, 242
 Tang::ComputedExpressionIteratorEnd, 255
 Tang::ComputedExpressionMap, 270
 Tang::ComputedExpressionNativeBoundFunction,
 284
 Tang::ComputedExpressionString, 298

—asCode
 Tang::ComputedExpression, 143
 Tang::ComputedExpressionArray, 157
 Tang::ComputedExpressionBoolean, 172
 Tang::ComputedExpressionCompiledFunction, 185
 Tang::ComputedExpressionError, 199
 Tang::ComputedExpressionFloat, 213
 Tang::ComputedExpressionInteger, 227
 Tang::ComputedExpressionIterator, 242
 Tang::ComputedExpressionIteratorEnd, 255
 Tang::ComputedExpressionMap, 270
 Tang::ComputedExpressionNativeBoundFunction,
 284
 Tang::ComputedExpressionString, 298

—assign_index
 Tang::ComputedExpression, 143
 Tang::ComputedExpressionArray, 157
 Tang::ComputedExpressionBoolean, 172
 Tang::ComputedExpressionCompiledFunction, 186
 Tang::ComputedExpressionError, 199
 Tang::ComputedExpressionFloat, 213
 Tang::ComputedExpressionInteger, 228
 Tang::ComputedExpressionIterator, 243
 Tang::ComputedExpressionIteratorEnd, 255
 Tang::ComputedExpressionMap, 271
 Tang::ComputedExpressionNativeBoundFunction,
 285
 Tang::ComputedExpressionString, 299

—boolean
 Tang::ComputedExpression, 144
 Tang::ComputedExpressionArray, 158
 Tang::ComputedExpressionBoolean, 173
 Tang::ComputedExpressionCompiledFunction, 186
 Tang::ComputedExpressionError, 199
 Tang::ComputedExpressionFloat, 214

—divide
 Tang::ComputedExpression, 144
 Tang::ComputedExpressionArray, 158
 Tang::ComputedExpressionBoolean, 173
 Tang::ComputedExpressionCompiledFunction, 186
 Tang::ComputedExpressionError, 200
 Tang::ComputedExpressionFloat, 214
 Tang::ComputedExpressionInteger, 228
 Tang::ComputedExpressionIterator, 243
 Tang::ComputedExpressionIteratorEnd, 256
 Tang::ComputedExpressionMap, 272
 Tang::ComputedExpressionNativeBoundFunction,
 285
 Tang::ComputedExpressionString, 299

—equal
 Tang::ComputedExpression, 145
 Tang::ComputedExpressionArray, 159
 Tang::ComputedExpressionBoolean, 173
 Tang::ComputedExpressionCompiledFunction, 187
 Tang::ComputedExpressionError, 200
 Tang::ComputedExpressionFloat, 215
 Tang::ComputedExpressionInteger, 229
 Tang::ComputedExpressionIterator, 244
 Tang::ComputedExpressionIteratorEnd, 257
 Tang::ComputedExpressionMap, 272
 Tang::ComputedExpressionNativeBoundFunction,
 286
 Tang::ComputedExpressionString, 300

—float
 Tang::ComputedExpression, 145
 Tang::ComputedExpressionArray, 159
 Tang::ComputedExpressionBoolean, 174
 Tang::ComputedExpressionCompiledFunction, 187
 Tang::ComputedExpressionError, 200
 Tang::ComputedExpressionFloat, 215
 Tang::ComputedExpressionInteger, 229
 Tang::ComputedExpressionIterator, 244
 Tang::ComputedExpressionIteratorEnd, 257
 Tang::ComputedExpressionMap, 273
 Tang::ComputedExpressionNativeBoundFunction,
 286
 Tang::ComputedExpressionString, 301

—`getIterator`
 Tang::ComputedExpression, 145
 Tang::ComputedExpressionArray, 159
 Tang::ComputedExpressionBoolean, 174
 Tang::ComputedExpressionCompiledFunction, 187
 Tang::ComputedExpressionError, 201
 Tang::ComputedExpressionFloat, 215
 Tang::ComputedExpressionInteger, 230
 Tang::ComputedExpressionIterator, 244
 Tang::ComputedExpressionIteratorEnd, 257
 Tang::ComputedExpressionMap, 273
 Tang::ComputedExpressionNativeBoundFunction, 287
 Tang::ComputedExpressionString, 301

—`index`
 Tang::ComputedExpression, 146
 Tang::ComputedExpressionArray, 160
 Tang::ComputedExpressionBoolean, 175
 Tang::ComputedExpressionCompiledFunction, 188
 Tang::ComputedExpressionError, 201
 Tang::ComputedExpressionFloat, 216
 Tang::ComputedExpressionInteger, 230
 Tang::ComputedExpressionIterator, 245
 Tang::ComputedExpressionIteratorEnd, 258
 Tang::ComputedExpressionMap, 273
 Tang::ComputedExpressionNativeBoundFunction, 287
 Tang::ComputedExpressionString, 301

—`integer`
 Tang::ComputedExpression, 146
 Tang::ComputedExpressionArray, 160
 Tang::ComputedExpressionBoolean, 175
 Tang::ComputedExpressionCompiledFunction, 188
 Tang::ComputedExpressionError, 201
 Tang::ComputedExpressionFloat, 216
 Tang::ComputedExpressionInteger, 230
 Tang::ComputedExpressionIterator, 245
 Tang::ComputedExpressionIteratorEnd, 258
 Tang::ComputedExpressionMap, 274
 Tang::ComputedExpressionNativeBoundFunction, 287
 Tang::ComputedExpressionString, 302

—`iteratorNext`
 Tang::ComputedExpression, 146
 Tang::ComputedExpressionArray, 160
 Tang::ComputedExpressionBoolean, 175
 Tang::ComputedExpressionCompiledFunction, 188
 Tang::ComputedExpressionError, 201
 Tang::ComputedExpressionFloat, 216
 Tang::ComputedExpressionInteger, 231
 Tang::ComputedExpressionIterator, 245
 Tang::ComputedExpressionIteratorEnd, 258
 Tang::ComputedExpressionMap, 274
 Tang::ComputedExpressionNativeBoundFunction, 288
 Tang::ComputedExpressionString, 302

—`lessThan`
 Tang::ComputedExpression, 147

—`modulo`
 Tang::ComputedExpression, 147
 Tang::ComputedExpressionArray, 161
 Tang::ComputedExpressionBoolean, 176
 Tang::ComputedExpressionCompiledFunction, 189
 Tang::ComputedExpressionError, 202
 Tang::ComputedExpressionFloat, 217
 Tang::ComputedExpressionInteger, 232
 Tang::ComputedExpressionIterator, 246
 Tang::ComputedExpressionIteratorEnd, 259
 Tang::ComputedExpressionMap, 275
 Tang::ComputedExpressionNativeBoundFunction, 288
 Tang::ComputedExpressionString, 303

—`multiply`
 Tang::ComputedExpression, 147
 Tang::ComputedExpressionArray, 162
 Tang::ComputedExpressionBoolean, 176
 Tang::ComputedExpressionCompiledFunction, 189
 Tang::ComputedExpressionError, 203
 Tang::ComputedExpressionFloat, 218
 Tang::ComputedExpressionInteger, 232
 Tang::ComputedExpressionIterator, 247
 Tang::ComputedExpressionIteratorEnd, 259
 Tang::ComputedExpressionMap, 275
 Tang::ComputedExpressionNativeBoundFunction, 289
 Tang::ComputedExpressionString, 304

—`negative`
 Tang::ComputedExpression, 148
 Tang::ComputedExpressionArray, 162
 Tang::ComputedExpressionBoolean, 177
 Tang::ComputedExpressionCompiledFunction, 190
 Tang::ComputedExpressionError, 203
 Tang::ComputedExpressionFloat, 218
 Tang::ComputedExpressionInteger, 233
 Tang::ComputedExpressionIterator, 247
 Tang::ComputedExpressionIteratorEnd, 260
 Tang::ComputedExpressionMap, 276
 Tang::ComputedExpressionNativeBoundFunction, 289
 Tang::ComputedExpressionString, 304

—`not`
 Tang::ComputedExpression, 148
 Tang::ComputedExpressionArray, 162
 Tang::ComputedExpressionBoolean, 177

Tang::ComputedExpressionCompiledFunction, 190
Tang::ComputedExpressionError, 203
Tang::ComputedExpressionFloat, 218
Tang::ComputedExpressionInteger, 233
Tang::ComputedExpressionIterator, 247
Tang::ComputedExpressionIteratorEnd, 260
Tang::ComputedExpressionMap, 276
Tang::ComputedExpressionNativeBoundFunction,
 289
Tang::ComputedExpressionString, 304
—period
 Tang::ComputedExpression, 148
 Tang::ComputedExpressionArray, 162
 Tang::ComputedExpressionBoolean, 177
 Tang::ComputedExpressionCompiledFunction, 190
 Tang::ComputedExpressionError, 203
 Tang::ComputedExpressionFloat, 219
 Tang::ComputedExpressionInteger, 233
 Tang::ComputedExpressionIterator, 247
 Tang::ComputedExpressionIteratorEnd, 260
 Tang::ComputedExpressionMap, 276
 Tang::ComputedExpressionNativeBoundFunction,
 290
 Tang::ComputedExpressionString, 305
—slice
 Tang::ComputedExpression, 149
 Tang::ComputedExpressionArray, 163
 Tang::ComputedExpressionBoolean, 178
 Tang::ComputedExpressionCompiledFunction, 191
 Tang::ComputedExpressionError, 204
 Tang::ComputedExpressionFloat, 219
 Tang::ComputedExpressionInteger, 234
 Tang::ComputedExpressionIterator, 248
 Tang::ComputedExpressionIteratorEnd, 261
 Tang::ComputedExpressionMap, 277
 Tang::ComputedExpressionNativeBoundFunction,
 290
 Tang::ComputedExpressionString, 305
—string
 Tang::ComputedExpression, 149
 Tang::ComputedExpressionArray, 163
 Tang::ComputedExpressionBoolean, 178
 Tang::ComputedExpressionCompiledFunction, 191
 Tang::ComputedExpressionError, 204
 Tang::ComputedExpressionFloat, 220
 Tang::ComputedExpressionInteger, 234
 Tang::ComputedExpressionIterator, 248
 Tang::ComputedExpressionIteratorEnd, 261
 Tang::ComputedExpressionMap, 277
 Tang::ComputedExpressionNativeBoundFunction,
 291
 Tang::ComputedExpressionString, 306
—subtract
 Tang::ComputedExpression, 149
 Tang::ComputedExpressionArray, 164
 Tang::ComputedExpressionBoolean, 178
 Tang::ComputedExpressionCompiledFunction, 191
 Tang::ComputedExpressionError, 204
Tang::ComputedExpressionFloat, 220
Tang::ComputedExpressionInteger, 235
Tang::ComputedExpressionIterator, 249
Tang::ComputedExpressionIteratorEnd, 261
Tang::ComputedExpressionMap, 277
Tang::ComputedExpressionNativeBoundFunction,
 291
Tang::ComputedExpressionString, 306
~GarbageCollected
 Tang::GarbageCollected, 316
ADD
 opcode.hpp, 410
Add
 Tang::AstNodeBinary, 32
addBreak
 Tang::Program, 340
addBytecode
 Tang::Program, 341
addContinue
 Tang::Program, 341
addIdentifier
 Tang::Program, 341
addIdentifierAssigned
 Tang::Program, 342
addString
 Tang::Program, 342
And
 Tang::AstNodeBinary, 32
append
 Tang::ComputedExpressionArray, 164
ARRAY
 opcode.hpp, 410
ASSIGNINDEX
 opcode.hpp, 410
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

AstNodeMap
 Tang::AstNodeMap, 97

AstNodePeriod
 Tang::AstNodePeriod, 101

AstNodePrint
 Tang::AstNodePrint, 106

AstNodeRangedFor
 Tang::AstNodeRangedFor, 110

AstNodeReturn
 Tang::AstNodeReturn, 114

AstNodeSlice
 Tang::AstNodeSlice, 119

AstNodeString
 Tang::AstNodeString, 123

AstNodeTernary
 Tang::AstNodeTernary, 128

AstNodeUnary
 Tang::AstNodeUnary, 133

AstNodeWhile
 Tang::AstNodeWhile, 138

BOOLEAN
 opcode.hpp, 410

Boolean
 Tang::AstNodeCast, 50

build/generated/location.hh, 363

bytesLength
 Tang::UnicodeString, 359

CALLFUNC
 opcode.hpp, 411

CASTBOOLEAN
 opcode.hpp, 411

CASTFLOAT
 opcode.hpp, 411

CASTINTEGER
 opcode.hpp, 411

CASTSTRING
 opcode.hpp, 411

CodeType
 Tang::Program, 340

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::AstNodeMap, 97
 Tang::AstNodePeriod, 102
 Tang::AstNodePrint, 106
 Tang::AstNodeRangedFor, 110
 Tang::AstNodeReturn, 115
 Tang::AstNodeSlice, 119
 Tang::AstNodeString, 124
 Tang::AstNodeTernary, 129
 Tang::AstNodeUnary, 133
 Tang::AstNodeWhile, 139

compileLiteral
 Tang::AstNodeString, 124

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::AstNodeMap, 98
 Tang::AstNodePeriod, 102
 Tang::AstNodePrint, 107
 Tang::AstNodeRangedFor, 111
 Tang::AstNodeReturn, 115
 Tang::AstNodeSlice, 120
 Tang::AstNodeString, 125
 Tang::AstNodeTernary, 129
 Tang::AstNodeUnary, 135
 Tang::AstNodeWhile, 139

compileScript
 Tang::TangBase, 352

ComputedExpressionArray
 Tang::ComputedExpressionArray, 156

ComputedExpressionBoolean
 Tang::ComputedExpressionBoolean, 171

ComputedExpressionCompiledFunction
 Tang::ComputedExpressionCompiledFunction, 185

ComputedExpressionError
 Tang::ComputedExpressionError, 198

ComputedExpressionFloat
 Tang::ComputedExpressionFloat, 212

ComputedExpressionInteger
 Tang::ComputedExpressionInteger, 227

ComputedExpressionIterator
 Tang::ComputedExpressionIterator, 242

ComputedExpressionMap
 Tang::ComputedExpressionMap, 270

ComputedExpressionNativeBoundFunction
 Tang::ComputedExpressionNativeBoundFunction, 284

ComputedExpressionString
 Tang::ComputedExpressionString, 297

COPY
 opcode.hpp, 410

currentIndex
 Tang::SingletonObjectPool< T >, 350

currentRecycledIndex
 Tang::SingletonObjectPool< T >, 350

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::AstNodeMap, 97
 Tang::AstNodePeriod, 101
 Tang::AstNodePrint, 106
 Tang::AstNodeRangedFor, 109
 Tang::AstNodeReturn, 114
 Tang::AstNodeSlice, 118
 Tang::AstNodeString, 123
 Tang::AstNodeTernary, 128
 Tang::AstNodeUnary, 133
 Tang::AstNodeWhile, 138

DIVIDE
 opcode.hpp, 410

Divide
 Tang::AstNodeBinary, 32

dump
 Tang::AstNode, 20
 Tang::AstNodeArray, 25
 Tang::AstNodeAssign, 29
 Tang::AstNodeBinary, 34
 Tang::AstNodeBlock, 39
 Tang::AstNodeBoolean, 42
 Tang::AstNodeBreak, 46
 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::AstNodeMap, 98
 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::AstNodeWhile, 140
 Tang::ComputedExpression, 150
 Tang::ComputedExpressionArray, 165
 Tang::ComputedExpressionBoolean, 179
 Tang::ComputedExpressionCompiledFunction, 192
 Tang::ComputedExpressionError, 206
 Tang::ComputedExpressionFloat, 221
 Tang::ComputedExpressionInteger, 235
 Tang::ComputedExpressionIterator, 249
 Tang::ComputedExpressionIteratorEnd, 263
 Tang::ComputedExpressionMap, 278
 Tang::ComputedExpressionNativeBoundFunction, 291
 Tang::ComputedExpressionString, 307

dumpBytecode
 Tang::Program, 342

DUMPPROGRAMCHECK
 program-dumpBytecode.cpp, 448

EQ
 opcode.hpp, 410

Equal
 Tang::AstNodeBinary, 32

Error
 Tang::Error, 312

error.cpp
 operator<<, 446

execute
 Tang::Program, 342

EXECUTEPROGRAMCHECK
 program-execute.cpp, 449

FLOAT
 opcode.hpp, 410

Float
 Tang::AstNodeCast, 50

FUNCTION
 opcode.hpp, 410

functionsDeclared
 Tang::Program, 347

GarbageCollected
 Tang::GarbageCollected, 316

garbageCollected.cpp
 operator<<, 447

get
 Tang::SingletonObjectPool< T >, 349

get_next_token
 Tang::HtmlEscape, 332
 Tang::HtmlEscapeAscii, 334
 Tang::TangScanner, 355
 Tang::Unescape, 357

getAst
 Tang::Program, 343

getBytecode
 Tang::Program, 343

getCode
 Tang::Program, 343

getCollection
 Tang::AstNodeIndex, 90

getContents
 Tang::ComputedExpressionArray, 165

getIdentifiers
 Tang::Program, 344

getIdentifiersAssigned
 Tang::Program, 344

getIndex
 Tang::AstNodeIndex, 90

getInstance
 Tang::SingletonObjectPool< T >, 349

GETITERATOR
 opcode.hpp, 411

getMethods
 Tang::ComputedExpressionArray, 165
 Tang::ComputedExpressionString, 307

getResult
 Tang::Program, 344

getStrings
 Tang::Program, 344

getValue
 Tang::ComputedExpressionFloat, 221
 Tang::ComputedExpressionInteger, 236
 Tang::ComputedExpressionString, 307

GreaterThan
 Tang::AstNodeBinary, 32

GreaterThanOrEqual
 Tang::AstNodeBinary, 32

GT
 opcode.hpp, 410

GTE
 opcode.hpp, 410

HtmlEscape
 Tang::HtmlEscape, 331

htmlEscape
 unicodeString.hpp, 418

HtmlEscapeAscii
 Tang::HtmlEscapeAscii, 333

htmlEscapeAscii
 unicodeString.hpp, 418

include/astNode.hpp, 365
include/astNodeArray.hpp, 366
include/astNodeAssign.hpp, 367
include/astNodeBinary.hpp, 368
include/astNodeBlock.hpp, 369
include/astNodeBoolean.hpp, 370
include/astNodeBreak.hpp, 371
include/astNodeCast.hpp, 372
include/astNodeContinue.hpp, 373
include/astNodeDoWhile.hpp, 374
include/astNodeFloat.hpp, 375
include/astNodeFor.hpp, 376
include/astNodeFunctionCall.hpp, 377
include/astNodeFunctionDeclaration.hpp, 378
include/astNodeIdentifier.hpp, 379
include/astNodeIfElse.hpp, 380
include/astNodeIndex.hpp, 381
include/astNodeInteger.hpp, 382
include/astNodeMap.hpp, 383
include/astNodePeriod.hpp, 384
include/astNodePrint.hpp, 385
include/astNodeRangedFor.hpp, 386
include/astNodeReturn.hpp, 387
include/astNodeSlice.hpp, 388
include/astNodeString.hpp, 389
include/astNodeTernary.hpp, 390
include/astNodeUnary.hpp, 391
include/astNodeWhile.hpp, 392
include/computedExpression.hpp, 393
include/computedExpressionArray.hpp, 394
include/computedExpressionBoolean.hpp, 395
include/computedExpressionCompiledFunction.hpp, 395
include/computedExpressionError.hpp, 396
include/computedExpressionFloat.hpp, 397
include/computedExpressionInteger.hpp, 398
include/computedExpressionIterator.hpp, 399
include/computedExpressionIteratorEnd.hpp, 400
include/computedExpressionMap.hpp, 401
include/computedExpressionNativeBoundFunction.hpp, 402
include/computedExpressionString.hpp, 403
include/error.hpp, 404
include/garbageCollected.hpp, 405
include/htmlEscape.hpp, 406
include/htmlEscapeAscii.hpp, 407
include/macros.hpp, 408
include/opcode.hpp, 409
include/program.hpp, 411
include singletonObjectPool.hpp, 412

include/tang.hpp, 413
include/tangBase.hpp, 414
include/tangScanner.hpp, 415
include/unescape.hpp, 416
include/unicodeString.hpp, 417
INDEX
 opcode.hpp, 410
INTEGER
 opcode.hpp, 410
Integer
 Tang::AstNodeCast, 50
is_equal
 Tang::ComputedExpression, 150–152
 Tang::ComputedExpressionArray, 166–168
 Tang::ComputedExpressionBoolean, 179–181
 Tang::ComputedExpressionCompiledFunction, 192–194
 Tang::ComputedExpressionError, 206, 208, 209
 Tang::ComputedExpressionFloat, 221–223
 Tang::ComputedExpressionInteger, 236–238
 Tang::ComputedExpressionIterator, 249–251
 Tang::ComputedExpressionIteratorEnd, 263, 265, 266
 Tang::ComputedExpressionMap, 278–280
 Tang::ComputedExpressionNativeBoundFunction, 291–293
 Tang::ComputedExpressionString, 308–310
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::AstNodeMap, 97
 Tang::AstNodePeriod, 101
 Tang::AstNodePrint, 106
 Tang::AstNodeRangedFor, 109
 Tang::AstNodeReturn, 114
 Tang::AstNodeSlice, 118
 Tang::AstNodeString, 123
 Tang::AstNodeTernary, 128
 Tang::AstNodeUnary, 133
 Tang::AstNodeWhile, 138
isCopyNeeded
 Tang::ComputedExpression, 152
Tang::ComputedExpressionArray, 168
Tang::ComputedExpressionBoolean, 181
Tang::ComputedExpressionCompiledFunction, 194
Tang::ComputedExpressionError, 209
Tang::ComputedExpressionFloat, 223
Tang::ComputedExpressionInteger, 238
Tang::ComputedExpressionIterator, 252
Tang::ComputedExpressionIteratorEnd, 266
Tang::ComputedExpressionMap, 280
Tang::ComputedExpressionNativeBoundFunction, 294
Tang::ComputedExpressionString, 310
Tang::GarbageCollected, 317
ISITERATOREND
 opcode.hpp, 411
ITERATORNEXT
 opcode.hpp, 411
JMP
 opcode.hpp, 410
JMPF
 opcode.hpp, 410
JMPF_POP
 opcode.hpp, 410
JMPT
 opcode.hpp, 410
JMPT_POP
 opcode.hpp, 410
length
 Tang::UnicodeString, 359
LessThan
 Tang::AstNodeBinary, 32
LessThanEqual
 Tang::AstNodeBinary, 32
location.hh
 operator<<, 364, 365
LT
 opcode.hpp, 410
LTE
 opcode.hpp, 410
make
 Tang::GarbageCollected, 317
make_shared
 Tang::TangBase, 353
makeCopy
 Tang::ComputedExpression, 153
 Tang::ComputedExpressionArray, 168
 Tang::ComputedExpressionBoolean, 182
 Tang::ComputedExpressionCompiledFunction, 195
 Tang::ComputedExpressionError, 209
 Tang::ComputedExpressionFloat, 224
 Tang::ComputedExpressionInteger, 238
 Tang::ComputedExpressionIterator, 252
 Tang::ComputedExpressionIteratorEnd, 266
 Tang::ComputedExpressionMap, 281
 Tang::ComputedExpressionNativeBoundFunction, 294

Tang::ComputedExpressionString, 310
 Tang::GarbageCollected, 318
MAP
 opcode.hpp, 410
MODULO
 opcode.hpp, 410
Modulo
 Tang::AstNodeBinary, 32
MULTIPLY
 opcode.hpp, 410
Multiply
 Tang::AstNodeBinary, 32
NEGATIVE
 opcode.hpp, 410
Negative
 Tang::AstNodeUnary, 133
NEQ
 opcode.hpp, 410
NOT
 opcode.hpp, 410
Not
 Tang::AstNodeUnary, 133
NotEqual
 Tang::AstNodeBinary, 32
NULLVAL
 opcode.hpp, 410
Opcode
 opcode.hpp, 410
opcode.hpp
 ADD, 410
 ARRAY, 410
 ASSIGNINDEX, 410
 BOOLEAN, 410
 CALLFUNC, 411
 CASTBOOLEAN, 411
 CASTFLOAT, 411
 CASTINTEGER, 411
 CASTSTRING, 411
 COPY, 410
 DIVIDE, 410
 EQ, 410
 FLOAT, 410
 FUNCTION, 410
 GETITERATOR, 411
 GT, 410
 GTE, 410
 INDEX, 410
 INTEGER, 410
 ISITERATOREND, 411
 ITERATORNEXT, 411
 JMP, 410
 JMPF, 410
 JMPF_POP, 410
 JMPT, 410
 JMPT_POP, 410
 LT, 410
 LTE, 410
MAP, 410
MODULO, 410
MULTIPLY, 410
NEGATIVE, 410
NEQ, 410
NOT, 410
NULLVAL, 410
Opcode, 410
PEEK, 410
PERIOD, 410
POKE, 410
POP, 410
PRINT, 411
RETURN, 411
SLICE, 411
STRING, 410
SUBTRACT, 410
Operation
 Tang::AstNodeBinary, 32
Operator
 Tang::AstNodeUnary, 132
operator std::string
 Tang::UnicodeString, 360
operator!
 Tang::GarbageCollected, 318
operator!=
 Tang::GarbageCollected, 319
operator<
 Tang::GarbageCollected, 323
 Tang::UnicodeString, 361
operator<<
 error.cpp, 446
 garbageCollected.cpp, 447
 location.hh, 364, 365
 Tang::Error, 313
 Tang::GarbageCollected, 330
operator<=
 Tang::GarbageCollected, 324
operator>
 Tang::GarbageCollected, 329
operator>=
 Tang::GarbageCollected, 329
operator*
 Tang::GarbageCollected, 320
operator+
 Tang::GarbageCollected, 321
 Tang::UnicodeString, 360
operator-
 Tang::GarbageCollected, 321, 322
operator->
 Tang::GarbageCollected, 322
operator/
 Tang::GarbageCollected, 323
operator=
 Tang::GarbageCollected, 324
operator==
 Tang::GarbageCollected, 326–328
 Tang::UnicodeString, 361

operator%
 Tang::GarbageCollected, 319

Or
 Tang::AstNodeBinary, 32

PEEK
 opcode.hpp, 410

PERIOD
 opcode.hpp, 410

POKE
 opcode.hpp, 410

POP
 opcode.hpp, 410

popBreakStack
 Tang::Program, 345

popContinueStack
 Tang::Program, 345

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::AstNodeMap, 96
 Tang::AstNodePeriod, 101
 Tang::AstNodePrint, 105
 Tang::AstNodeRangedFor, 109
 Tang::AstNodeReturn, 114
 Tang::AstNodeSlice, 118
 Tang::AstNodeString, 123
 Tang::AstNodeTernary, 128
 Tang::AstNodeUnary, 133
 Tang::AstNodeWhile, 138

PRINT
 opcode.hpp, 411

Program
 Tang::Program, 340

program-dumpBytecode.cpp
 DUMPPROGRAMCHECK, 448

program-execute.cpp
 EXECUTEPROGRAMCHECK, 449
 STACKCHECK, 450

pushEnvironment
 Tang::Program, 346

recycle

 Tang::SingletonObjectPool< T >, 350

RETURN
 opcode.hpp, 411

Script
 Tang::Program, 340

setFunctionStackDeclaration
 Tang::Program, 346

setJumpTarget
 Tang::Program, 347

SLICE
 opcode.hpp, 411
 src/astNode.cpp, 420
 src/astNodeArray.cpp, 420
 src/astNodeAssign.cpp, 421
 src/astNodeBinary.cpp, 421
 src/astNodeBlock.cpp, 422
 src/astNodeBoolean.cpp, 423
 src/astNodeBreak.cpp, 423
 src/astNodeCast.cpp, 424
 src/astNodeContinue.cpp, 425
 src/astNodeDoWhile.cpp, 425
 src/astNodeFloat.cpp, 426
 src/astNodeFor.cpp, 427
 src/astNodeFunctionCall.cpp, 427
 src/astNodeFunctionDeclaration.cpp, 428
 src/astNodeIdentifier.cpp, 429
 src/astNodeIfElse.cpp, 429
 src/astNodeIndex.cpp, 430
 src/astNodeInteger.cpp, 431
 src/astNodeMap.cpp, 431
 src/astNodePeriod.cpp, 432
 src/astNodePrint.cpp, 433
 src/astNodeRangedFor.cpp, 433
 src/astNodeReturn.cpp, 434
 src/astNodeSlice.cpp, 435
 src/astNodeString.cpp, 435
 src/astNodeTernary.cpp, 436
 src/astNodeUnary.cpp, 437
 src/astNodeWhile.cpp, 437
 src/computedExpression.cpp, 438
 src/computedExpressionArray.cpp, 439
 src/computedExpressionBoolean.cpp, 439
 src/computedExpressionCompiledFunction.cpp, 440
 src/computedExpressionError.cpp, 441
 src/computedExpressionFloat.cpp, 441
 src/computedExpressionInteger.cpp, 442
 src/computedExpressionIterator.cpp, 443
 src/computedExpressionIteratorEnd.cpp, 443
 src/computedExpressionMap.cpp, 444
 src/computedExpressionNativeBoundFunction.cpp, 445
 src/computedExpressionString.cpp, 445
 src/error.cpp, 446
 src/garbageCollected.cpp, 447
 src/program-dumpBytecode.cpp, 448
 src/program-execute.cpp, 449
 src/program.cpp, 450
 src/tangBase.cpp, 451
 src/unicodeString.cpp, 452

STACKCHECK
 program-execute.cpp, 450

STRING
 opcode.hpp, 410

String
 Tang::AstNodeCast, 50

substr
 Tang::UnicodeString, 361

SUBTRACT
 opcode.hpp, 410

Subtract
 Tang::AstNodeBinary, 32

Tang::AstNode, 15
 AstNode, 18
 compile, 19
 compilePreprocess, 19
 Default, 18
 dump, 20
 IsAssignment, 18
 PreprocessState, 18

Tang::AstNodeArray, 20
 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
 GreaterThanEqual, 32
 IsAssignment, 33
 LessThan, 32
 LessThanEqual, 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, 42
 IsAssignment, 41
 PreprocessState, 41

Tang::AstNodeBreak, 43
 AstNodeBreak, 45
 compile, 45
 compilePreprocess, 46
 Default, 45
 dump, 46
 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::AstNodeMap, 95
AstNodeMap, 97
compile, 97
compilePreprocess, 98
Default, 97
dump, 98
IsAssignment, 97
PreprocessState, 96
Tang::AstNodePeriod, 99
AstNodePeriod, 101
compile, 102
compilePreprocess, 102
Default, 101
dump, 103
IsAssignment, 101
PreprocessState, 101
Tang::AstNodePrint, 103
AstNodePrint, 106
compile, 106
compilePreprocess, 107
Default, 106
dump, 107
IsAssignment, 106
PreprocessState, 105
Type, 106
Tang::AstNodeRangedFor, 108
AstNodeRangedFor, 110
compile, 110
compilePreprocess, 111
Default, 109
dump, 112
IsAssignment, 109
PreprocessState, 109
Tang::AstNodeReturn, 112
AstNodeReturn, 114
compile, 115
compilePreprocess, 115
Default, 114
dump, 116
IsAssignment, 114
PreprocessState, 114
Tang::AstNodeSlice, 116
AstNodeSlice, 119
compile, 119
compilePreprocess, 120
Default, 118
dump, 120
IsAssignment, 118
PreprocessState, 118
Tang::AstNodeString, 121
AstNodeString, 123
compile, 124
compileLiteral, 124
compilePreprocess, 125
Default, 123

dump, 125
 IsAssignment, 123
 PreprocessState, 123
 Tang::AstNodeTernary, 126
 AstNodeTernary, 128
 compile, 129
 compilePreprocess, 129
 Default, 128
 dump, 130
 IsAssignment, 128
 PreprocessState, 128
 Tang::AstNodeUnary, 130
 AstNodeUnary, 133
 compile, 133
 compilePreprocess, 135
 Default, 133
 dump, 135
 IsAssignment, 133
 Negative, 133
 Not, 133
 Operator, 132
 PreprocessState, 133
 Tang::AstNodeWhile, 136
 AstNodeWhile, 138
 compile, 139
 compilePreprocess, 139
 Default, 138
 dump, 140
 IsAssignment, 138
 PreprocessState, 138
 Tang::ComputedExpression, 140
 __add, 143
 __asCode, 143
 __assign_index, 143
 __boolean, 144
 __divide, 144
 __equal, 145
 __float, 145
 __getIterator, 145
 __index, 146
 __integer, 146
 __iteratorNext, 146
 __lessThan, 147
 __modulo, 147
 __multiply, 147
 __negative, 148
 __not, 148
 __period, 148
 __slice, 149
 __string, 149
 __subtract, 149
 dump, 150
 is_equal, 150–152
 isCopyNeeded, 152
 makeCopy, 153
 Tang::ComputedExpressionArray, 153
 __add, 157
 __asCode, 157
 __assign_index, 157
 __boolean, 158
 __divide, 158
 __equal, 159
 __float, 159
 __getIterator, 159
 __index, 160
 __integer, 160
 __iteratorNext, 160
 __lessThan, 161
 __modulo, 161
 __multiply, 162
 __negative, 162
 __not, 162
 __period, 162
 __slice, 163
 __string, 163
 __subtract, 164
 append, 164
 ComputedExpressionArray, 156
 dump, 165
 getContents, 165
 getMethods, 165
 is_equal, 166–168
 isCopyNeeded, 168
 makeCopy, 168
 Tang::ComputedExpressionBoolean, 169
 __add, 172
 __asCode, 172
 __assign_index, 172
 __boolean, 173
 __divide, 173
 __equal, 173
 __float, 174
 __getIterator, 174
 __index, 175
 __integer, 175
 __iteratorNext, 175
 __lessThan, 176
 __modulo, 176
 __multiply, 176
 __negative, 177
 __not, 177
 __period, 177
 __slice, 178
 __string, 178
 __subtract, 178
 ComputedExpressionBoolean, 171
 dump, 179
 is_equal, 179–181
 isCopyNeeded, 181
 makeCopy, 182
 Tang::ComputedExpressionCompiledFunction, 182
 __add, 185
 __asCode, 185
 __assign_index, 186
 __boolean, 186
 __divide, 186

__equal, 187
 __float, 187
 __getIterator, 187
 __index, 188
 __integer, 188
 __iteratorNext, 188
 __lessThan, 189
 __modulo, 189
 __multiply, 189
 __negative, 190
 __not, 190
 __period, 190
 __slice, 191
 __string, 191
 __subtract, 191
 ComputedExpressionCompiledFunction, 185
 dump, 192
 is_equal, 192–194
 isCopyNeeded, 194
 makeCopy, 195
Tang::ComputedExpressionError, 195
 __add, 198
 __asCode, 199
 __assign_index, 199
 __boolean, 199
 __divide, 200
 __equal, 200
 __float, 200
 __getIterator, 201
 __index, 201
 __integer, 201
 __iteratorNext, 201
 __lessThan, 202
 __modulo, 202
 __multiply, 203
 __negative, 203
 __not, 203
 __period, 203
 __slice, 204
 __string, 204
 __subtract, 204
 ComputedExpressionError, 198
 dump, 206
 is_equal, 206, 208, 209
 isCopyNeeded, 209
 makeCopy, 209
Tang::ComputedExpressionFloat, 210
 __add, 213
 __asCode, 213
 __assign_index, 213
 __boolean, 214
 __divide, 214
 __equal, 215
 __float, 215
 __getIterator, 215
 __index, 216
 __integer, 216
 __iteratorNext, 216
 __lessThan, 217
 __modulo, 217
 __multiply, 218
 __negative, 218
 __not, 218
 __period, 219
 __slice, 219
 __string, 220
 __subtract, 220
 ComputedExpressionFloat, 212
 dump, 221
 getValue, 221
 is_equal, 221–223
 isCopyNeeded, 223
 makeCopy, 224
Tang::ComputedExpressionInteger, 224
 __add, 227
 __asCode, 227
 __assign_index, 228
 __boolean, 228
 __divide, 228
 __equal, 229
 __float, 229
 __getIterator, 230
 __index, 230
 __integer, 230
 __iteratorNext, 231
 __lessThan, 231
 __modulo, 232
 __multiply, 232
 __negative, 233
 __not, 233
 __period, 233
 __slice, 234
 __string, 234
 __subtract, 235
 ComputedExpressionInteger, 227
 dump, 235
 getValue, 236
 is_equal, 236–238
 isCopyNeeded, 238
 makeCopy, 238
Tang::ComputedExpressionIterator, 239
 __add, 242
 __asCode, 242
 __assign_index, 243
 __boolean, 243
 __divide, 243
 __equal, 244
 __float, 244
 __getIterator, 244
 __index, 245
 __integer, 245
 __iteratorNext, 245
 __lessThan, 246
 __modulo, 246
 __multiply, 247
 __negative, 247

```

__not, 247
__period, 247
__slice, 248
__string, 248
__subtract, 249
ComputedExpressionIterator, 242
dump, 249
is_equal, 249–251
isCopyNeeded, 252
makeCopy, 252
Tang::ComputedExpressionIteratorEnd, 253
__add, 255
__asCode, 255
__assign_index, 255
__boolean, 256
__divide, 256
__equal, 257
__float, 257
__getIterator, 257
__index, 258
__integer, 258
__iteratorNext, 258
__lessThan, 259
__modulo, 259
__multiply, 259
__negative, 260
__not, 260
__period, 260
__slice, 261
__string, 261
__subtract, 261
dump, 263
is_equal, 263, 265, 266
isCopyNeeded, 266
makeCopy, 266
Tang::ComputedExpressionMap, 267
__add, 270
__asCode, 270
__assign_index, 271
__boolean, 271
__divide, 272
__equal, 272
__float, 273
__getIterator, 273
__index, 273
__integer, 274
__iteratorNext, 274
__lessThan, 274
__modulo, 275
__multiply, 275
__negative, 276
__not, 276
__period, 276
__slice, 277
__string, 277
__subtract, 277
ComputedExpressionMap, 270
dump, 278
is_equal, 278–280
isCopyNeeded, 280
makeCopy, 281
Tang::ComputedExpressionNativeBoundFunction, 281
__add, 284
__asCode, 284
__assign_index, 285
__boolean, 285
__divide, 285
__equal, 286
__float, 286
__getIterator, 287
__index, 287
__integer, 287
__iteratorNext, 288
__lessThan, 288
__modulo, 288
__multiply, 289
__negative, 289
__not, 289
__period, 290
__slice, 290
__string, 291
__subtract, 291
ComputedExpressionNativeBoundFunction, 284
dump, 291
is_equal, 291–293
isCopyNeeded, 294
makeCopy, 294
Tang::ComputedExpressionString, 295
__add, 298
__asCode, 298
__assign_index, 299
__boolean, 299
__divide, 300
__equal, 300
__float, 301
__getIterator, 301
__index, 301
__integer, 302
__iteratorNext, 302
__lessThan, 303
__modulo, 303
__multiply, 304
__negative, 304
__not, 304
__period, 305
__slice, 305
__string, 306
__subtract, 306
ComputedExpressionString, 297
dump, 307
getMethods, 307
getValue, 307
is_equal, 308–310
isCopyNeeded, 310
makeCopy, 310
Tang::Error, 311

```

Error, 312
operator<<, 313
Tang::GarbageCollected, 313
~GarbageCollected, 316
GarbageCollected, 316
isCopyNeeded, 317
make, 317
makeCopy, 318
operator!, 318
operator!=, 319
operator<, 323
operator<<, 330
operator<=, 324
operator>, 329
operator>=, 329
operator*, 320
operator+, 321
operator-, 321, 322
operator->, 322
operator/, 323
operator=, 324
operator==, 326–328
operator%, 319
Tang::HtmlEscape, 330
get_next_token, 332
HtmlEscape, 331
Tang::HtmlEscapeAscii, 332
get_next_token, 334
HtmlEscapeAscii, 333
Tang::location, 334
Tang::position, 336
Tang::Program, 337
addBreak, 340
addBytecode, 341
addContinue, 341
addIdentifier, 341
addIdentifierAssigned, 342
addString, 342
CodeType, 340
dumpBytecode, 342
execute, 342
functionsDeclared, 347
getAst, 343
getBytecode, 343
getCode, 343
getIdentifiers, 344
getIdentifiersAssigned, 344
getResult, 344
getStrings, 344
popBreakStack, 345
popContinueStack, 345
Program, 340
pushEnvironment, 346
Script, 340
setFunctionStackDeclaration, 346
setJumpTarget, 347
Template, 340
Tang::SingletonObjectPool< T >, 348
currentIndex, 350
currentRecycledIndex, 350
get, 349
getInstance, 349
recycle, 350
Tang::TangBase, 351
compileScript, 352
make_shared, 353
TangBase, 352
Tang::TangScanner, 353
get_next_token, 355
TangScanner, 355
Tang::Unescape, 356
get_next_token, 357
Unescape, 357
Tang::UnicodeString, 358
bytesLength, 359
length, 359
operator std::string, 360
operator<, 361
operator+, 360
operator==, 361
substr, 361
UnicodeString, 359
TangBase
Tang::TangBase, 352
TangScanner
Tang::TangScanner, 355
Template
Tang::Program, 340
test/test.cpp, 452
test/testGarbageCollected.cpp, 454
test/testSingletonObjectPool.cpp, 454
test/testUnicodeString.cpp, 455
Type
Tang::AstNodeCast, 49
Tang::AstNodePrint, 106
Unescape
Tang::Unescape, 357
unescape
unicodeString.hpp, 419
UnicodeString
Tang::UnicodeString, 359
unicodeString.hpp
htmlEscape, 418
htmlEscapeAscii, 418
unescape, 419