

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	7
4.1 File List	7
5 Class Documentation	11
5.1 Tang::AstNode Class Reference	11
5.1.1 Detailed Description	13
5.1.2 Constructor & Destructor Documentation	13
5.1.2.1 AstNode()	13
5.1.3 Member Function Documentation	13
5.1.3.1 compile()	13
5.1.3.2 compilePreprocess()	14
5.1.3.3 dump()	14
5.2 Tang::AstNodeArray Class Reference	15
5.2.1 Detailed Description	16
5.2.2 Constructor & Destructor Documentation	16
5.2.2.1 AstNodeArray()	16
5.2.3 Member Function Documentation	17
5.2.3.1 compile()	17
5.2.3.2 compilePreprocess()	17
5.2.3.3 dump()	17
5.3 Tang::AstNodeAssign Class Reference	18
5.3.1 Detailed Description	19
5.3.2 Constructor & Destructor Documentation	19
5.3.2.1 AstNodeAssign()	19
5.3.3 Member Function Documentation	19
5.3.3.1 compile()	19
5.3.3.2 compilePreprocess()	20
5.3.3.3 dump()	20
5.4 Tang::AstNodeBinary Class Reference	21
5.4.1 Detailed Description	22
5.4.2 Member Enumeration Documentation	22
5.4.2.1 Operation	22
5.4.3 Constructor & Destructor Documentation	23

5.4.3.1 AstNodeBinary()	23
5.4.4 Member Function Documentation	23
5.4.4.1 compile()	23
5.4.4.2 compilePreprocess()	24
5.4.4.3 dump()	24
5.5 Tang::AstNodeBlock Class Reference	25
5.5.1 Detailed Description	26
5.5.2 Constructor & Destructor Documentation	26
5.5.2.1 AstNodeBlock()	26
5.5.3 Member Function Documentation	26
5.5.3.1 compile()	26
5.5.3.2 compilePreprocess()	27
5.5.3.3 dump()	27
5.6 Tang::AstNodeBoolean Class Reference	28
5.6.1 Detailed Description	28
5.6.2 Constructor & Destructor Documentation	29
5.6.2.1 AstNodeBoolean()	29
5.6.3 Member Function Documentation	29
5.6.3.1 compile()	29
5.6.3.2 compilePreprocess()	30
5.6.3.3 dump()	30
5.7 Tang::AstNodeBreak Class Reference	30
5.7.1 Detailed Description	31
5.7.2 Constructor & Destructor Documentation	31
5.7.2.1 AstNodeBreak()	32
5.7.3 Member Function Documentation	32
5.7.3.1 compile()	32
5.7.3.2 compilePreprocess()	33
5.7.3.3 dump()	33
5.8 Tang::AstNodeCast Class Reference	33
5.8.1 Detailed Description	34
5.8.2 Member Enumeration Documentation	35
5.8.2.1 Type	35
5.8.3 Constructor & Destructor Documentation	35
5.8.3.1 AstNodeCast()	35
5.8.4 Member Function Documentation	35
5.8.4.1 compile()	35
5.8.4.2 compilePreprocess()	36
5.8.4.3 dump()	36
5.9 Tang::AstNodeContinue Class Reference	37
5.9.1 Detailed Description	38
5.9.2 Constructor & Destructor Documentation	38

5.9.2.1 AstNodeContinue()	38
5.9.3 Member Function Documentation	38
5.9.3.1 compile()	38
5.9.3.2 compilePreprocess()	39
5.9.3.3 dump()	39
5.10 Tang::AstNodeDoWhile Class Reference	40
5.10.1 Detailed Description	41
5.10.2 Constructor & Destructor Documentation	41
5.10.2.1 AstNodeDoWhile()	41
5.10.3 Member Function Documentation	41
5.10.3.1 compile()	41
5.10.3.2 compilePreprocess()	42
5.10.3.3 dump()	42
5.11 Tang::AstNodeFloat Class Reference	43
5.11.1 Detailed Description	44
5.11.2 Constructor & Destructor Documentation	44
5.11.2.1 AstNodeFloat()	44
5.11.3 Member Function Documentation	44
5.11.3.1 compile()	44
5.11.3.2 compilePreprocess()	45
5.11.3.3 dump()	45
5.12 Tang::AstNodeFor Class Reference	46
5.12.1 Detailed Description	47
5.12.2 Constructor & Destructor Documentation	47
5.12.2.1 AstNodeFor()	47
5.12.3 Member Function Documentation	47
5.12.3.1 compile()	47
5.12.3.2 compilePreprocess()	48
5.12.3.3 dump()	48
5.13 Tang::AstNodeFunctionCall Class Reference	49
5.13.1 Detailed Description	50
5.13.2 Constructor & Destructor Documentation	50
5.13.2.1 AstNodeFunctionCall()	50
5.13.3 Member Function Documentation	50
5.13.3.1 compile()	50
5.13.3.2 compilePreprocess()	51
5.13.3.3 dump()	51
5.14 Tang::AstNodeFunctionDeclaration Class Reference	52
5.14.1 Detailed Description	53
5.14.2 Constructor & Destructor Documentation	53
5.14.2.1 AstNodeFunctionDeclaration()	53
5.14.3 Member Function Documentation	53

5.14.3.1 compile()	53
5.14.3.2 compilePreprocess()	54
5.14.3.3 dump()	54
5.15 Tang::AstNodeIdentifier Class Reference	56
5.15.1 Detailed Description	57
5.15.2 Constructor & Destructor Documentation	57
5.15.2.1 AstNodeIdentifier()	57
5.15.3 Member Function Documentation	58
5.15.3.1 compile()	58
5.15.3.2 compilePreprocess()	58
5.15.3.3 dump()	59
5.16 Tang::AstNodeIfElse Class Reference	59
5.16.1 Detailed Description	60
5.16.2 Constructor & Destructor Documentation	61
5.16.2.1 AstNodeIfElse() [1/2]	61
5.16.2.2 AstNodeIfElse() [2/2]	61
5.16.3 Member Function Documentation	61
5.16.3.1 compile()	61
5.16.3.2 compilePreprocess()	62
5.16.3.3 dump()	62
5.17 Tang::AstNodeIndex Class Reference	63
5.17.1 Detailed Description	64
5.17.2 Constructor & Destructor Documentation	64
5.17.2.1 AstNodeIndex()	64
5.17.3 Member Function Documentation	64
5.17.3.1 compile()	64
5.17.3.2 compilePreprocess()	65
5.17.3.3 dump()	65
5.18 Tang::AstNodeInteger Class Reference	66
5.18.1 Detailed Description	67
5.18.2 Constructor & Destructor Documentation	67
5.18.2.1 AstNodeInteger()	67
5.18.3 Member Function Documentation	67
5.18.3.1 compile()	67
5.18.3.2 compilePreprocess()	68
5.18.3.3 dump()	68
5.19 Tang::AstNodePrint Class Reference	69
5.19.1 Detailed Description	70
5.19.2 Member Enumeration Documentation	70
5.19.2.1 Type	70
5.19.3 Constructor & Destructor Documentation	70
5.19.3.1 AstNodePrint()	70

5.19.4 Member Function Documentation	71
5.19.4.1 compile()	71
5.19.4.2 compilePreprocess()	71
5.19.4.3 dump()	71
5.20 Tang::AstNodeReturn Class Reference	72
5.20.1 Detailed Description	73
5.20.2 Constructor & Destructor Documentation	73
5.20.2.1 AstNodeReturn()	73
5.20.3 Member Function Documentation	73
5.20.3.1 compile()	73
5.20.3.2 compilePreprocess()	74
5.20.3.3 dump()	74
5.21 Tang::AstNodeString Class Reference	75
5.21.1 Detailed Description	76
5.21.2 Constructor & Destructor Documentation	76
5.21.2.1 AstNodeString()	76
5.21.3 Member Function Documentation	76
5.21.3.1 compile()	76
5.21.3.2 compileLiteral()	77
5.21.3.3 compilePreprocess()	78
5.21.3.4 dump()	78
5.22 Tang::AstNodeTernary Class Reference	79
5.22.1 Detailed Description	80
5.22.2 Constructor & Destructor Documentation	80
5.22.2.1 AstNodeTernary()	80
5.22.3 Member Function Documentation	80
5.22.3.1 compile()	80
5.22.3.2 compilePreprocess()	81
5.22.3.3 dump()	81
5.23 Tang::AstNodeUnary Class Reference	82
5.23.1 Detailed Description	83
5.23.2 Member Enumeration Documentation	83
5.23.2.1 Operator	83
5.23.3 Constructor & Destructor Documentation	83
5.23.3.1 AstNodeUnary()	83
5.23.4 Member Function Documentation	84
5.23.4.1 compile()	84
5.23.4.2 compilePreprocess()	84
5.23.4.3 dump()	84
5.24 Tang::AstNodeWhile Class Reference	85
5.24.1 Detailed Description	86
5.24.2 Constructor & Destructor Documentation	86

5.24.2.1 AstNodeWhile()	86
5.24.3 Member Function Documentation	86
5.24.3.1 compile()	86
5.24.3.2 compilePreprocess()	87
5.24.3.3 dump()	87
5.25 Tang::ComputedExpression Class Reference	88
5.25.1 Detailed Description	90
5.25.2 Member Function Documentation	90
5.25.2.1 __add()	90
5.25.2.2 __boolean()	90
5.25.2.3 __divide()	90
5.25.2.4 __equal()	91
5.25.2.5 __float()	91
5.25.2.6 __index()	91
5.25.2.7 __integer()	92
5.25.2.8 __lessThan()	92
5.25.2.9 __modulo()	93
5.25.2.10 __multiply()	93
5.25.2.11 __negative()	93
5.25.2.12 __not()	94
5.25.2.13 __string()	94
5.25.2.14 __subtract()	94
5.25.2.15 dump()	95
5.25.2.16 is_equal() [1/6]	95
5.25.2.17 is_equal() [2/6]	95
5.25.2.18 is_equal() [3/6]	96
5.25.2.19 is_equal() [4/6]	96
5.25.2.20 is_equal() [5/6]	96
5.25.2.21 is_equal() [6/6]	97
5.25.2.22 makeCopy()	97
5.26 Tang::ComputedExpressionArray Class Reference	98
5.26.1 Detailed Description	99
5.26.2 Constructor & Destructor Documentation	99
5.26.2.1 ComputedExpressionArray()	99
5.26.3 Member Function Documentation	100
5.26.3.1 __add()	100
5.26.3.2 __boolean()	100
5.26.3.3 __divide()	100
5.26.3.4 __equal()	101
5.26.3.5 __float()	101
5.26.3.6 __index()	101
5.26.3.7 __integer()	102

5.26.3.8 __lessThan()	102
5.26.3.9 __modulo()	103
5.26.3.10 __multiply()	103
5.26.3.11 __negative()	103
5.26.3.12 __not()	104
5.26.3.13 __string()	104
5.26.3.14 __subtract()	104
5.26.3.15 dump()	105
5.26.3.16 is_equal() [1/6]	105
5.26.3.17 is_equal() [2/6]	105
5.26.3.18 is_equal() [3/6]	106
5.26.3.19 is_equal() [4/6]	106
5.26.3.20 is_equal() [5/6]	106
5.26.3.21 is_equal() [6/6]	107
5.26.3.22 makeCopy()	107
5.27 Tang::ComputedExpressionBoolean Class Reference	108
5.27.1 Detailed Description	109
5.27.2 Constructor & Destructor Documentation	109
5.27.2.1 ComputedExpressionBoolean()	109
5.27.3 Member Function Documentation	110
5.27.3.1 __add()	110
5.27.3.2 __boolean()	110
5.27.3.3 __divide()	110
5.27.3.4 __equal()	111
5.27.3.5 __float()	111
5.27.3.6 __index()	111
5.27.3.7 __integer()	112
5.27.3.8 __lessThan()	112
5.27.3.9 __modulo()	113
5.27.3.10 __multiply()	114
5.27.3.11 __negative()	114
5.27.3.12 __not()	115
5.27.3.13 __string()	115
5.27.3.14 __subtract()	115
5.27.3.15 dump()	116
5.27.3.16 is_equal() [1/6]	116
5.27.3.17 is_equal() [2/6]	116
5.27.3.18 is_equal() [3/6]	117
5.27.3.19 is_equal() [4/6]	117
5.27.3.20 is_equal() [5/6]	117
5.27.3.21 is_equal() [6/6]	118
5.27.3.22 makeCopy()	118

5.28 Tang::ComputedExpressionCompiledFunction Class Reference	119
5.28.1 Detailed Description	120
5.28.2 Constructor & Destructor Documentation	120
5.28.2.1 ComputedExpressionCompiledFunction()	121
5.28.3 Member Function Documentation	121
5.28.3.1 __add()	121
5.28.3.2 __boolean()	121
5.28.3.3 __divide()	122
5.28.3.4 __equal()	122
5.28.3.5 __float()	122
5.28.3.6 __index()	123
5.28.3.7 __integer()	123
5.28.3.8 __lessThan()	123
5.28.3.9 __modulo()	124
5.28.3.10 __multiply()	124
5.28.3.11 __negative()	124
5.28.3.12 __not()	125
5.28.3.13 __string()	125
5.28.3.14 __subtract()	125
5.28.3.15 dump()	126
5.28.3.16 is_equal() [1/6]	126
5.28.3.17 is_equal() [2/6]	126
5.28.3.18 is_equal() [3/6]	127
5.28.3.19 is_equal() [4/6]	127
5.28.3.20 is_equal() [5/6]	128
5.28.3.21 is_equal() [6/6]	128
5.28.3.22 makeCopy()	128
5.29 Tang::ComputedExpressionError Class Reference	129
5.29.1 Detailed Description	130
5.29.2 Constructor & Destructor Documentation	130
5.29.2.1 ComputedExpressionError()	130
5.29.3 Member Function Documentation	131
5.29.3.1 __add()	131
5.29.3.2 __boolean()	131
5.29.3.3 __divide()	131
5.29.3.4 __equal()	132
5.29.3.5 __float()	132
5.29.3.6 __index()	132
5.29.3.7 __integer()	133
5.29.3.8 __lessThan()	133
5.29.3.9 __modulo()	133
5.29.3.10 __multiply()	134

5.29.3.11 <code>__negative()</code>	134
5.29.3.12 <code>__not()</code>	135
5.29.3.13 <code>__string()</code>	135
5.29.3.14 <code>__subtract()</code>	135
5.29.3.15 <code>dump()</code>	136
5.29.3.16 <code>is_equal()</code> [1/6]	136
5.29.3.17 <code>is_equal()</code> [2/6]	136
5.29.3.18 <code>is_equal()</code> [3/6]	137
5.29.3.19 <code>is_equal()</code> [4/6]	137
5.29.3.20 <code>is_equal()</code> [5/6]	137
5.29.3.21 <code>is_equal()</code> [6/6]	138
5.29.3.22 <code>makeCopy()</code>	138
5.30 Tang::ComputedExpressionFloat Class Reference	139
5.30.1 Detailed Description	140
5.30.2 Constructor & Destructor Documentation	140
5.30.2.1 <code>ComputedExpressionFloat()</code>	141
5.30.3 Member Function Documentation	141
5.30.3.1 <code>__add()</code>	141
5.30.3.2 <code>__boolean()</code>	141
5.30.3.3 <code>__divide()</code>	141
5.30.3.4 <code>__equal()</code>	142
5.30.3.5 <code>__float()</code>	142
5.30.3.6 <code>__index()</code>	142
5.30.3.7 <code>__integer()</code>	143
5.30.3.8 <code>__lessThan()</code>	143
5.30.3.9 <code>__modulo()</code>	143
5.30.3.10 <code>__multiply()</code>	144
5.30.3.11 <code>__negative()</code>	144
5.30.3.12 <code>__not()</code>	145
5.30.3.13 <code>__string()</code>	145
5.30.3.14 <code>__subtract()</code>	145
5.30.3.15 <code>dump()</code>	146
5.30.3.16 <code>is_equal()</code> [1/6]	146
5.30.3.17 <code>is_equal()</code> [2/6]	146
5.30.3.18 <code>is_equal()</code> [3/6]	147
5.30.3.19 <code>is_equal()</code> [4/6]	147
5.30.3.20 <code>is_equal()</code> [5/6]	148
5.30.3.21 <code>is_equal()</code> [6/6]	148
5.30.3.22 <code>makeCopy()</code>	148
5.31 Tang::ComputedExpressionInteger Class Reference	149
5.31.1 Detailed Description	150
5.31.2 Constructor & Destructor Documentation	150

5.31.2.1 ComputedExpressionInteger()	151
5.31.3 Member Function Documentation	151
5.31.3.1 __add()	151
5.31.3.2 __boolean()	151
5.31.3.3 __divide()	151
5.31.3.4 __equal()	152
5.31.3.5 __float()	152
5.31.3.6 __index()	152
5.31.3.7 __integer()	153
5.31.3.8 __lessThan()	153
5.31.3.9 __modulo()	153
5.31.3.10 __multiply()	154
5.31.3.11 __negative()	154
5.31.3.12 __not()	155
5.31.3.13 __string()	155
5.31.3.14 __subtract()	155
5.31.3.15 dump()	156
5.31.3.16 is_equal() [1/6]	156
5.31.3.17 is_equal() [2/6]	156
5.31.3.18 is_equal() [3/6]	157
5.31.3.19 is_equal() [4/6]	157
5.31.3.20 is_equal() [5/6]	158
5.31.3.21 is_equal() [6/6]	158
5.31.3.22 makeCopy()	158
5.32 Tang::ComputedExpressionString Class Reference	159
5.32.1 Detailed Description	160
5.32.2 Constructor & Destructor Documentation	160
5.32.2.1 ComputedExpressionString()	160
5.32.3 Member Function Documentation	161
5.32.3.1 __add()	161
5.32.3.2 __boolean()	161
5.32.3.3 __divide()	161
5.32.3.4 __equal()	162
5.32.3.5 __float()	162
5.32.3.6 __index()	162
5.32.3.7 __integer()	163
5.32.3.8 __lessThan()	163
5.32.3.9 __modulo()	164
5.32.3.10 __multiply()	165
5.32.3.11 __negative()	165
5.32.3.12 __not()	166
5.32.3.13 __string()	166

5.32.3.14 __subtract()	166
5.32.3.15 dump()	167
5.32.3.16 is_equal() [1/6]	167
5.32.3.17 is_equal() [2/6]	167
5.32.3.18 is_equal() [3/6]	168
5.32.3.19 is_equal() [4/6]	168
5.32.3.20 is_equal() [5/6]	168
5.32.3.21 is_equal() [6/6]	169
5.32.3.22 makeCopy()	169
5.33 Tang::Error Class Reference	170
5.33.1 Detailed Description	171
5.33.2 Constructor & Destructor Documentation	171
5.33.2.1 Error() [1/2]	171
5.33.2.2 Error() [2/2]	171
5.33.3 Friends And Related Function Documentation	171
5.33.3.1 operator<<	172
5.34 Tang::GarbageCollected Class Reference	172
5.34.1 Detailed Description	174
5.34.2 Constructor & Destructor Documentation	174
5.34.2.1 GarbageCollected() [1/3]	174
5.34.2.2 GarbageCollected() [2/3]	175
5.34.2.3 ~GarbageCollected()	175
5.34.2.4 GarbageCollected() [3/3]	175
5.34.3 Member Function Documentation	175
5.34.3.1 make()	175
5.34.3.2 operator"!"()	176
5.34.3.3 operator"!="()	176
5.34.3.4 operator%()	177
5.34.3.5 operator*() [1/2]	178
5.34.3.6 operator*() [2/2]	178
5.34.3.7 operator+()	178
5.34.3.8 operator-() [1/2]	179
5.34.3.9 operator-() [2/2]	179
5.34.3.10 operator->()	180
5.34.3.11 operator/()	180
5.34.3.12 operator<()	181
5.34.3.13 operator<=()	181
5.34.3.14 operator=() [1/2]	182
5.34.3.15 operator=() [2/2]	182
5.34.3.16 operator==(1/8)	183
5.34.3.17 operator==(2/8)	183
5.34.3.18 operator==(3/8)	184

5.34.3.19 operator==() [4 / 8]	184
5.34.3.20 operator==() [5 / 8]	184
5.34.3.21 operator==() [6 / 8]	185
5.34.3.22 operator==() [7 / 8]	185
5.34.3.23 operator==() [8 / 8]	185
5.34.3.24 operator>()	187
5.34.3.25 operator>=()	187
5.34.4 Friends And Related Function Documentation	188
5.34.4.1 operator<<	188
5.35 Tang::location Class Reference	188
5.35.1 Detailed Description	190
5.36 Tang::position Class Reference	190
5.36.1 Detailed Description	191
5.37 Tang::Program Class Reference	191
5.37.1 Detailed Description	193
5.37.2 Member Enumeration Documentation	193
5.37.2.1 CodeType	193
5.37.3 Constructor & Destructor Documentation	193
5.37.3.1 Program()	193
5.37.4 Member Function Documentation	194
5.37.4.1 addBreak()	194
5.37.4.2 addBytecode()	194
5.37.4.3 addContinue()	194
5.37.4.4 addIdentifier()	195
5.37.4.5 addString()	195
5.37.4.6 dumpBytecode()	195
5.37.4.7 execute()	195
5.37.4.8 getAst()	196
5.37.4.9 getBytecode()	196
5.37.4.10 getCode()	196
5.37.4.11 getIdentifiers()	197
5.37.4.12 getResult()	197
5.37.4.13 getStrings()	197
5.37.4.14 popBreakStack()	197
5.37.4.15 popContinueStack()	198
5.37.4.16 pushEnvironment()	198
5.37.4.17 setFunctionStackDeclaration()	199
5.37.4.18 setJumpTarget()	199
5.37.5 Member Data Documentation	200
5.37.5.1 functionsDeclared	200
5.38 Tang::SingletonObjectPool< T > Class Template Reference	200
5.38.1 Detailed Description	200

5.38.2 Member Function Documentation	201
5.38.2.1 get()	201
5.38.2.2 getInstance()	201
5.38.2.3 recycle()	201
5.39 Tang::TangBase Class Reference	202
5.39.1 Detailed Description	202
5.39.2 Constructor & Destructor Documentation	202
5.39.2.1 TangBase()	202
5.39.3 Member Function Documentation	202
5.39.3.1 compileScript()	202
5.40 Tang::TangScanner Class Reference	203
5.40.1 Detailed Description	204
5.40.2 Constructor & Destructor Documentation	204
5.40.2.1 TangScanner()	204
5.40.3 Member Function Documentation	204
5.40.3.1 get_next_token()	204
6 File Documentation	207
6.1 build/generated/location.hh File Reference	207
6.1.1 Detailed Description	208
6.1.2 Function Documentation	208
6.1.2.1 operator<<() [1/2]	208
6.1.2.2 operator<<() [2/2]	209
6.2 include/astNode.hpp File Reference	209
6.2.1 Detailed Description	210
6.3 include/astNodeArray.hpp File Reference	210
6.3.1 Detailed Description	211
6.4 include/astNodeAssign.hpp File Reference	211
6.4.1 Detailed Description	212
6.5 include/astNodeBinary.hpp File Reference	212
6.5.1 Detailed Description	213
6.6 include/astNodeBlock.hpp File Reference	213
6.6.1 Detailed Description	214
6.7 include/astNodeBoolean.hpp File Reference	214
6.7.1 Detailed Description	215
6.8 include/astNodeBreak.hpp File Reference	215
6.8.1 Detailed Description	216
6.9 include/astNodeCast.hpp File Reference	216
6.9.1 Detailed Description	217
6.10 include/astNodeContinue.hpp File Reference	217
6.10.1 Detailed Description	218
6.11 include/astNodeDoWhile.hpp File Reference	218

6.11.1 Detailed Description	219
6.12 include/astNodeFloat.hpp File Reference	219
6.12.1 Detailed Description	220
6.13 include/astNodeFor.hpp File Reference	220
6.13.1 Detailed Description	221
6.14 include/astNodeFunctionCall.hpp File Reference	221
6.14.1 Detailed Description	222
6.15 include/astNodeFunctionDeclaration.hpp File Reference	222
6.15.1 Detailed Description	223
6.16 include/astNodeIdentifier.hpp File Reference	223
6.16.1 Detailed Description	224
6.17 include/astNodeIfElse.hpp File Reference	224
6.17.1 Detailed Description	225
6.18 include/astNodeIndex.hpp File Reference	225
6.18.1 Detailed Description	226
6.19 include/astNodeInteger.hpp File Reference	226
6.19.1 Detailed Description	227
6.20 include/astNodePrint.hpp File Reference	227
6.20.1 Detailed Description	228
6.21 include/astNodeReturn.hpp File Reference	228
6.21.1 Detailed Description	229
6.22 include/astNodeString.hpp File Reference	229
6.22.1 Detailed Description	230
6.23 include/astNodeTernary.hpp File Reference	230
6.23.1 Detailed Description	231
6.24 include/astNodeUnary.hpp File Reference	231
6.24.1 Detailed Description	232
6.25 include/astNodeWhile.hpp File Reference	232
6.25.1 Detailed Description	233
6.26 include/computedExpression.hpp File Reference	233
6.26.1 Detailed Description	233
6.27 include/computedExpressionArray.hpp File Reference	234
6.27.1 Detailed Description	234
6.28 include/computedExpressionBoolean.hpp File Reference	235
6.28.1 Detailed Description	235
6.29 include/computedExpressionCompiledFunction.hpp File Reference	236
6.29.1 Detailed Description	236
6.30 include/computedExpressionError.hpp File Reference	237
6.30.1 Detailed Description	237
6.31 include/computedExpressionFloat.hpp File Reference	238
6.31.1 Detailed Description	238
6.32 include/computedExpressionInteger.hpp File Reference	239

6.32.1 Detailed Description	239
6.33 include/computedExpressionString.hpp File Reference	240
6.33.1 Detailed Description	240
6.34 include/error.hpp File Reference	241
6.34.1 Detailed Description	241
6.35 include/garbageCollected.hpp File Reference	242
6.35.1 Detailed Description	242
6.36 include/macros.hpp File Reference	242
6.36.1 Detailed Description	243
6.37 include/opcode.hpp File Reference	243
6.37.1 Detailed Description	243
6.37.2 Enumeration Type Documentation	243
6.37.2.1 Opcode	243
6.38 include/program.hpp File Reference	244
6.38.1 Detailed Description	245
6.39 include/singletonObjectPool.hpp File Reference	245
6.39.1 Detailed Description	246
6.40 include/tang.hpp File Reference	246
6.40.1 Detailed Description	247
6.41 include/tangBase.hpp File Reference	247
6.41.1 Detailed Description	248
6.42 include/tangScanner.hpp File Reference	249
6.42.1 Detailed Description	250
6.43 src/astNode.cpp File Reference	250
6.43.1 Detailed Description	250
6.44 src/astNodeArray.cpp File Reference	250
6.44.1 Detailed Description	251
6.45 src/astNodeAssign.cpp File Reference	251
6.45.1 Detailed Description	252
6.46 src/astNodeBinary.cpp File Reference	252
6.46.1 Detailed Description	253
6.47 src/astNodeBlock.cpp File Reference	253
6.47.1 Detailed Description	253
6.48 src/astNodeBoolean.cpp File Reference	253
6.48.1 Detailed Description	254
6.49 src/astNodeBreak.cpp File Reference	254
6.49.1 Detailed Description	255
6.50 src/astNodeCast.cpp File Reference	255
6.50.1 Detailed Description	255
6.51 src/astNodeContinue.cpp File Reference	255
6.51.1 Detailed Description	256
6.52 src/astNodeDoWhile.cpp File Reference	256

6.52.1 Detailed Description	257
6.53 src/astNodeFloat.cpp File Reference	257
6.53.1 Detailed Description	258
6.54 src/astNodeFor.cpp File Reference	258
6.54.1 Detailed Description	258
6.55 src/astNodeFunctionCall.cpp File Reference	258
6.55.1 Detailed Description	259
6.56 src/astNodeFunctionDeclaration.cpp File Reference	259
6.56.1 Detailed Description	260
6.57 src/astNodeIdentifier.cpp File Reference	260
6.57.1 Detailed Description	261
6.58 src/astNodeIfElse.cpp File Reference	261
6.58.1 Detailed Description	261
6.59 src/astNodeIndex.cpp File Reference	261
6.59.1 Detailed Description	262
6.60 src/astNodeInteger.cpp File Reference	262
6.60.1 Detailed Description	263
6.61 src/astNodePrint.cpp File Reference	263
6.61.1 Detailed Description	263
6.62 src/astNodeReturn.cpp File Reference	263
6.62.1 Detailed Description	264
6.63 src/astNodeString.cpp File Reference	264
6.63.1 Detailed Description	265
6.64 src/astNodeTernary.cpp File Reference	265
6.64.1 Detailed Description	266
6.65 src/astNodeUnary.cpp File Reference	266
6.65.1 Detailed Description	266
6.66 src/astNodeWhile.cpp File Reference	266
6.66.1 Detailed Description	267
6.67 src/computedExpression.cpp File Reference	267
6.67.1 Detailed Description	268
6.68 src/computedExpressionArray.cpp File Reference	268
6.68.1 Detailed Description	269
6.69 src/computedExpressionBoolean.cpp File Reference	269
6.69.1 Detailed Description	269
6.70 src/computedExpressionCompiledFunction.cpp File Reference	269
6.70.1 Detailed Description	270
6.71 src/computedExpressionError.cpp File Reference	270
6.71.1 Detailed Description	271
6.72 src/computedExpressionFloat.cpp File Reference	271
6.72.1 Detailed Description	271
6.73 src/computedExpressionInteger.cpp File Reference	271

6.73.1 Detailed Description	272
6.74 src/computedExpressionString.cpp File Reference	272
6.74.1 Detailed Description	273
6.75 src/error.cpp File Reference	273
6.75.1 Detailed Description	273
6.75.2 Function Documentation	273
6.75.2.1 operator<<()	273
6.76 src/program-dumpBytecode.cpp File Reference	274
6.76.1 Detailed Description	274
6.76.2 Macro Definition Documentation	274
6.76.2.1 DUMPPROGRAMCHECK	275
6.77 src/program-execute.cpp File Reference	275
6.77.1 Detailed Description	276
6.77.2 Macro Definition Documentation	276
6.77.2.1 EXECUTEPROGRAMCHECK	276
6.77.2.2 STACKCHECK	276
6.78 src/program.cpp File Reference	276
6.78.1 Detailed Description	277
6.79 src/tangBase.cpp File Reference	277
6.79.1 Detailed Description	278
6.80 test/test.cpp File Reference	278
6.80.1 Detailed Description	279
6.81 test/testGarbageCollected.cpp File Reference	279
6.81.1 Detailed Description	280
6.82 test/testSingletonObjectPool.cpp File Reference	280
6.82.1 Detailed Description	280
Index	281

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	11
Tang::AstNodeArray	15
Tang::AstNodeAssign	18
Tang::AstNodeBinary	21
Tang::AstNodeBlock	25
Tang::AstNodeBoolean	28
Tang::AstNodeBreak	30
Tang::AstNodeCast	33
Tang::AstNodeContinue	37
Tang::AstNodeDoWhile	40
Tang::AstNodeFloat	43
Tang::AstNodeFor	46
Tang::AstNodeFunctionCall	49
Tang::AstNodeFunctionDeclaration	52
Tang::AstNodeIdentifier	56
Tang::AstNodeIfElse	59
Tang::AstNodeIndex	63
Tang::AstNodeInteger	66
Tang::AstNodePrint	69
Tang::AstNodeReturn	72
Tang::AstNodeString	75
Tang::AstNodeTernary	79
Tang::AstNodeUnary	82
Tang::AstNodeWhile	85
Tang::ComputedExpression	88
Tang::ComputedExpressionArray	98
Tang::ComputedExpressionBoolean	108
Tang::ComputedExpressionCompiledFunction	119
Tang::ComputedExpressionError	129
Tang::ComputedExpressionFloat	139
Tang::ComputedExpressionInteger	149
Tang::ComputedExpressionString	159
Tang::Error	170
Tang::GarbageCollected	172
Tang::location	188

Tang::position	190
Tang::Program	191
Tang::SingletonObjectPool< T >	200
Tang::TangBase	202
TangTangFlexLexer	
Tang::TangScanner	203

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)	11
Tang::AstNodeArray	An AstNode that represents an array literal	15
Tang::AstNodeAssign	An AstNode that represents a binary expression	18
Tang::AstNodeBinary	An AstNode that represents a binary expression	21
Tang::AstNodeBlock	An AstNode that represents a code block	25
Tang::AstNodeBoolean	An AstNode that represents a boolean literal	28
Tang::AstNodeBreak	An AstNode that represents a <code>break</code> statement	30
Tang::AstNodeCast	An AstNode that represents a typecast of an expression	33
Tang::AstNodeContinue	An AstNode that represents a <code>continue</code> statement	37
Tang::AstNodeDoWhile	An AstNode that represents a <code>do..while</code> statement	40
Tang::AstNodeFloat	An AstNode that represents an float literal	43
Tang::AstNodeFor	An AstNode that represents an <code>if()</code> statement	46
Tang::AstNodeFunctionCall	An AstNode that represents a function call	49
Tang::AstNodeFunctionDeclaration	An AstNode that represents a function declaration	52
Tang::AstNodeIdentifier	An AstNode that represents an identifier	56
Tang::AstNodeIfElse	An AstNode that represents an <code>if..else</code> statement	59
Tang::AstNodeIndex	An AstNode that represents an index into a collection	63
Tang::AstNodeInteger	An AstNode that represents an integer literal	66

Tang::AstNodePrint	
An AstNode that represents a print typeoperation	69
Tang::AstNodeReturn	
An AstNode that represents a <code>return</code> statement	72
Tang::AstNodeString	
An AstNode that represents a string literal	75
Tang::AstNodeTernary	
An AstNode that represents a ternary expression	79
Tang::AstNodeUnary	
An AstNode that represents a unary negation	82
Tang::AstNodeWhile	
An AstNode that represents a while statement	85
Tang::ComputedExpression	
Represents the result of a computation that has been executed	88
Tang::ComputedExpressionArray	
Represents an Array that is the result of a computation	98
Tang::ComputedExpressionBoolean	
Represents an Boolean that is the result of a computation	108
Tang::ComputedExpressionCompiledFunction	
Represents a Compiled Function declared in the script	119
Tang::ComputedExpressionError	
Represents a Runtime Error	129
Tang::ComputedExpressionFloat	
Represents a Float that is the result of a computation	139
Tang::ComputedExpressionInteger	
Represents an Integer that is the result of a computation	149
Tang::ComputedExpressionString	
Represents a String that is the result of a computation	159
Tang::Error	
Used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error	170
Tang::GarbageCollected	
A container that acts as a resource-counting garbage collector for the specified type	172
Tang::location	
Two points in a source file	188
Tang::position	
A point in a source file	190
Tang::Program	
Represents a compiled script or template that may be executed	191
Tang::SingletonObjectPool< T >	
A thread-safe, singleton object pool of the designated type	200
Tang::TangBase	
The base class for the Tang programming language	202
Tang::TangScanner	
The Flex lexer class for the main Tang language	203

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 <code>Tang::location</code> class	207
include/astNode.hpp	
Declare the <code>Tang::AstNode</code> base class	209
include/astNodeArray.hpp	
Declare the <code>Tang::AstNodeArray</code> class	210
include/astNodeAssign.hpp	
Declare the <code>Tang::AstNodeAssign</code> class	211
include/astNodeBinary.hpp	
Declare the <code>Tang::AstNodeBinary</code> class	212
include/astNodeBlock.hpp	
Declare the <code>Tang::AstNodeBlock</code> class	213
include/astNodeBoolean.hpp	
Declare the <code>Tang::AstNodeBoolean</code> class	214
include/astNodeBreak.hpp	
Declare the <code>Tang::AstNodeBreak</code> class	215
include/astNodeCast.hpp	
Declare the <code>Tang::AstNodeCast</code> class	216
include/astNodeContinue.hpp	
Declare the <code>Tang::AstNodeContinue</code> class	217
include/astNodeDoWhile.hpp	
Declare the <code>Tang::AstNodeDoWhile</code> class	218
include/astNodeFloat.hpp	
Declare the <code>Tang::AstNodeFloat</code> class	219
include/astNodeFor.hpp	
Declare the <code>Tang::AstNodeFor</code> class	220
include/astNodeFunctionCall.hpp	
Declare the <code>Tang::AstNodeFunctionCall</code> class	221
include/astNodeFunctionDeclaration.hpp	
Declare the <code>Tang::AstNodeFunctionDeclaration</code> class	222
include/astNodeIdentifier.hpp	
Declare the <code>Tang::AstNodeIdentifier</code> class	223
include/astNodeIfElse.hpp	
Declare the <code>Tang::AstNodeIfElse</code> class	224
include/astNodeIndex.hpp	
Declare the <code>Tang::AstNodeIndex</code> class	225

include/ astNodeInteger.hpp	Declare the Tang::AstNodeInteger class	226
include/ astNodePrint.hpp	Declare the Tang::AstNodePrint class	227
include/ astNodeReturn.hpp	Declare the Tang::AstNodeReturn class	228
include/ astNodeString.hpp	Declare the Tang::AstNodeString class	229
include/ astNodeTernary.hpp	Declare the Tang::AstNodeTernary class	230
include/ astNodeUnary.hpp	Declare the Tang::AstNodeUnary class	231
include/ astNodeWhile.hpp	Declare the Tang::AstNodeWhile class	232
include/ computedExpression.hpp	Declare the Tang::ComputedExpression base class	233
include/ computedExpressionArray.hpp	Declare the Tang::ComputedExpressionArray class	234
include/ computedExpressionBoolean.hpp	Declare the Tang::ComputedExpressionBoolean class	235
include/ computedExpressionCompiledFunction.hpp	Declare the Tang::ComputedExpressionCompiledFunction class	236
include/ computedExpressionError.hpp	Declare the Tang::ComputedExpressionError class	237
include/ computedExpressionFloat.hpp	Declare the Tang::ComputedExpressionFloat class	238
include/ computedExpressionInteger.hpp	Declare the Tang::ComputedExpressionInteger class	239
include/ computedExpressionString.hpp	Declare the Tang::ComputedExpressionString class	240
include/ error.hpp	Declare the Tang::Error class used to describe syntax and runtime errors	241
include/ garbageCollected.hpp	Declare the Tang::GarbageCollected class	242
include/ macros.hpp	Contains generic macros	242
include/ opcode.hpp	Declare the Opcodes used in the Bytecode representation of a program	243
include/ program.hpp	Declare the Tang::Program class used to compile and execute source code	244
include/ singletonObjectPool.hpp	Declare the Tang::SingletonObjectPool class	245
include/ tang.hpp	Header file supplied for use by 3rd party code so that they can easily include all necessary headers	246
include/ tangBase.hpp	Declare the Tang::TangBase class used to interact with Tang	247
include/ tangScanner.hpp	Declare the Tang::TangScanner used to tokenize a Tang script	249
src/ astNode.cpp	Define the Tang::AstNode class	250
src/ astNodeArray.cpp	Define the Tang::AstNodeArray class	250
src/ astNodeAssign.cpp	Define the Tang::AstNodeAssign class	251
src/ astNodeBinary.cpp	Define the Tang::AstNodeBinary class	252

src/ astNodeBlock.cpp	
Define the Tang::AstNodeBlock class	253
src/ astNodeBoolean.cpp	
Define the Tang::AstNodeBoolean class	253
src/ astNodeBreak.cpp	
Define the Tang::AstNodeBreak class	254
src/ astNodeCast.cpp	
Define the Tang::AstNodeCast class	255
src/ astNodeContinue.cpp	
Define the Tang::AstNodeContinue class	255
src/ astNodeDoWhile.cpp	
Define the Tang::AstNodeDoWhile class	256
src/ astNodeFloat.cpp	
Define the Tang::AstNodeFloat class	257
src/ astNodeFor.cpp	
Define the Tang::AstNodeFor class	258
src/ astNodeFunctionCall.cpp	
Define the Tang::AstNodeFunctionCall class	258
src/ astNodeFunctionDeclaration.cpp	
Define the Tang::AstNodeFunctionDeclaration class	259
src/ astNodeIdentifier.cpp	
Define the Tang::AstNodeIdentifier class	260
src/ astNodeIfElse.cpp	
Define the Tang::AstNodeIfElse class	261
src/ astNodeIndex.cpp	
Define the Tang::AstNodeIndex class	261
src/ astNodeInteger.cpp	
Define the Tang::AstNodeInteger class	262
src/ astNodePrint.cpp	
Define the Tang::AstNodePrint class	263
src/ astNodeReturn.cpp	
Define the Tang::AstNodeReturn class	263
src/ astNodeString.cpp	
Define the Tang::AstNodeString class	264
src/ astNodeTernary.cpp	
Define the Tang::AstNodeTernary class	265
src/ astNodeUnary.cpp	
Define the Tang::AstNodeUnary class	266
src/ astNodeWhile.cpp	
Define the Tang::AstNodeWhile class	266
src/ computedExpression.cpp	
Define the Tang::ComputedExpression class	267
src/ computedExpressionArray.cpp	
Define the Tang::ComputedExpressionArray class	268
src/ computedExpressionBoolean.cpp	
Define the Tang::ComputedExpressionBoolean class	269
src/ computedExpressionCompiledFunction.cpp	
Define the Tang::ComputedExpressionCompiledFunction class	269
src/ computedExpressionError.cpp	
Define the Tang::ComputedExpressionError class	270
src/ computedExpressionFloat.cpp	
Define the Tang::ComputedExpressionFloat class	271
src/ computedExpressionInteger.cpp	
Define the Tang::ComputedExpressionInteger class	271
src/ computedExpressionString.cpp	
Define the Tang::ComputedExpressionString class	272
src/ error.cpp	
Define the Tang::Error class	273

src/ program-dumpBytecode.cpp	
Define the Tang::Program::dumpBytecode method	274
src/ program-execute.cpp	
Define the Tang::Program::execute method	275
src/ program.cpp	
Define the Tang::Program class	276
src/ tangBase.cpp	
Define the Tang::TangBase class	277
test/ test.cpp	
Test the general language behaviors	278
test/ testGarbageCollected.cpp	
Test the generic behavior of the Tang::GarbageCollected class	279
test/ testSingletonObjectPool.cpp	
Test the generic behavior of the Tang::SingletonObjectPool class	280

Chapter 5

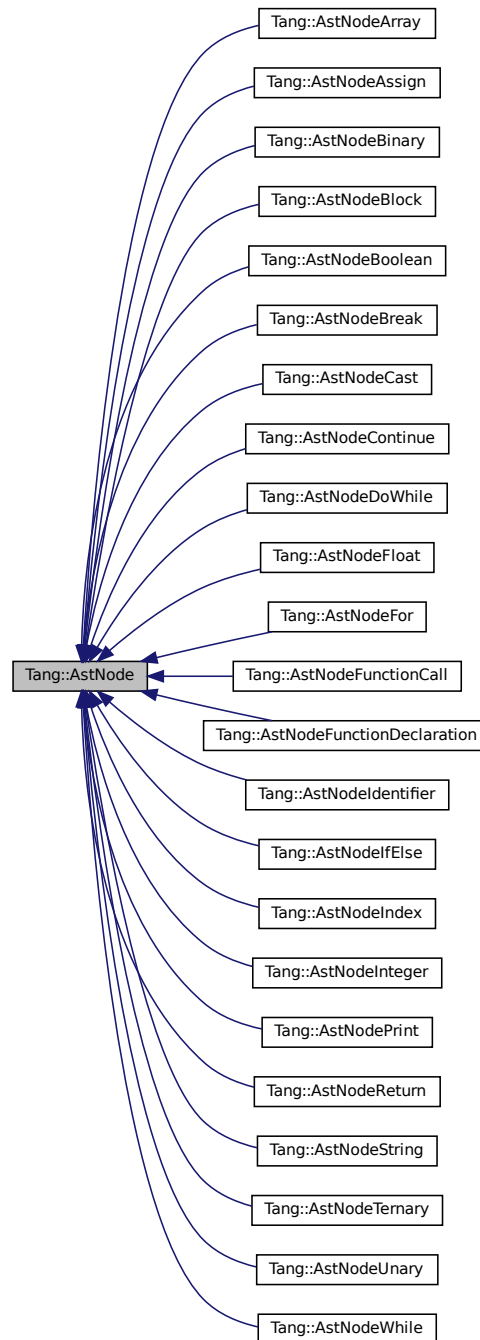
Class Documentation

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:



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) const

Run any preprocess analysis needed before compilation.

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 Constructor & Destructor Documentation

5.1.2.1 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.
-----------------------	---

5.1.3 Member Function Documentation

5.1.3.1 compile()

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

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

Parameters

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

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeReturn](#), [Tang::AstNodePrint](#), [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:



5.1.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeReturn](#), [Tang::AstNodePrint](#), [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](#).

5.1.3.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeReturn](#), [Tang::AstNodePrint](#), [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:

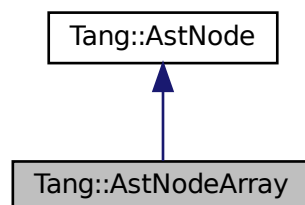
- [include/astNode.hpp](#)
- [src/astNode.cpp](#)

5.2 Tang::AstNodeArray Class Reference

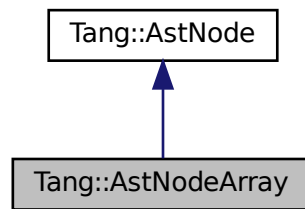
An [AstNode](#) that represents an array literal.

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

Inheritance diagram for `Tang::AstNodeArray`:



Collaboration diagram for Tang::AstNodeArray:



Public 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) const override
Run any preprocess analysis needed before compilation.

5.2.1 Detailed Description

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

5.2.2 Constructor & Destructor Documentation

5.2.2.1 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.

5.2.3 Member Function Documentation

5.2.3.1 compile()

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

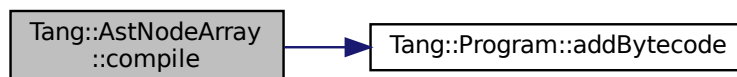
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.2.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.2.3.3 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:

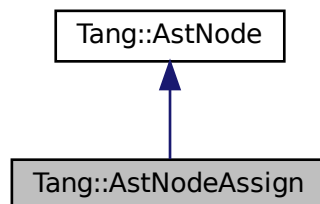
- [include/astNodeArray.hpp](#)
- [src/astNodeArray.cpp](#)

5.3 Tang::AstNodeAssign Class Reference

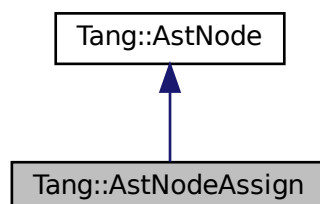
An [AstNode](#) that represents a binary expression.

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

Inheritance diagram for Tang::AstNodeAssign:



Collaboration diagram for Tang::AstNodeAssign:



Public 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) const override
Run any preprocess analysis needed before compilation.

5.3.1 Detailed Description

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

5.3.2 Constructor & Destructor Documentation

5.3.2.1 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.

5.3.3 Member Function Documentation

5.3.3.1 compile()

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

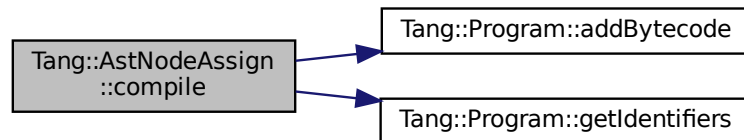
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.3.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.3.3.3 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:

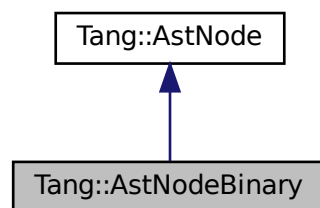
- include/[astNodeAssign.hpp](#)
- src/[astNodeAssign.cpp](#)

5.4 Tang::AstNodeBinary Class Reference

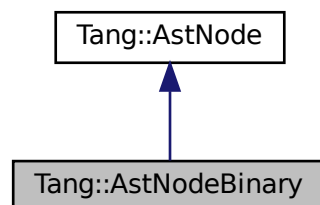
An [AstNode](#) that represents a binary expression.

```
#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.

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) const override
Run any preprocess analysis needed before compilation.

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.
NotEqual	Indicates lhs != rhs.
And	Indicates lhs && rhs with short-circuit evaluation.
Or	Indicates lhs rhs with short-circuit evaluation.

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]
```

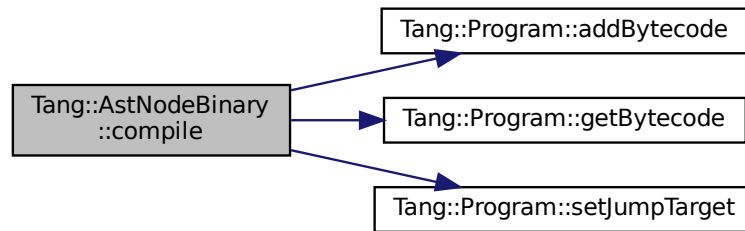
Compile the ast of the provided [Tang::Program](#).

Parameters

<i>program</i>	The 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 ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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:

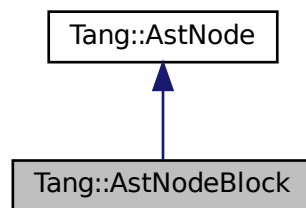
- [include/astNodeBinary.hpp](#)
- [src/astNodeBinary.cpp](#)

5.5 Tang::AstNodeBlock Class Reference

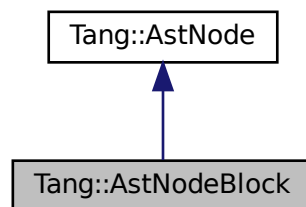
An [AstNode](#) that represents a code block.

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

Inheritance diagram for Tang::AstNodeBlock:



Collaboration diagram for Tang::AstNodeBlock:



Public 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) const override
Run any preprocess analysis needed before compilation.

5.5.1 Detailed Description

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

5.5.2 Constructor & Destructor Documentation

5.5.2.1 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.

5.5.3 Member Function Documentation

5.5.3.1 compile()

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

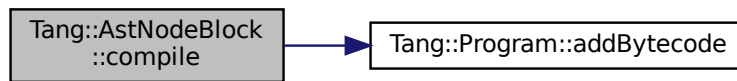
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.5.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.5.3.3 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:

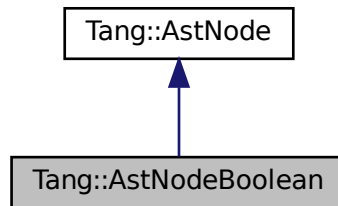
- [include/astNodeBlock.hpp](#)
- [src/astNodeBlock.cpp](#)

5.6 Tang::AstNodeBoolean Class Reference

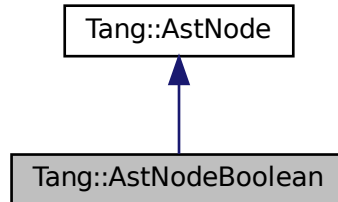
An [AstNode](#) that represents a boolean literal.

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

Inheritance diagram for Tang::AstNodeBoolean:



Collaboration diagram for Tang::AstNodeBoolean:



Public 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) const
Run any preprocess analysis needed before compilation.

5.6.1 Detailed Description

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

5.6.2 Constructor & Destructor Documentation

5.6.2.1 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.

5.6.3 Member Function Documentation

5.6.3.1 compile()

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

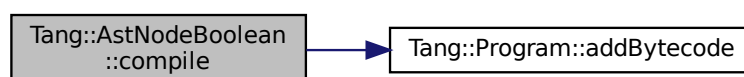
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.6.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeReturn](#), [Tang::AstNodePrint](#), [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](#).

5.6.3.3 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:

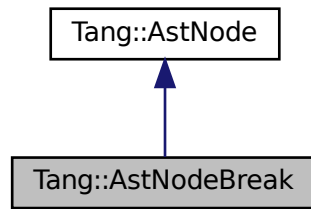
- [include/astNodeBoolean.hpp](#)
- [src/astNodeBoolean.cpp](#)

5.7 Tang::AstNodeBreak Class Reference

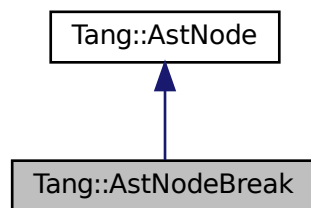
An [AstNode](#) that represents a `break` statement.

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

Inheritance diagram for Tang::AstNodeBreak:



Collaboration diagram for Tang::AstNodeBreak:



Public 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) const
Run any preprocess analysis needed before compilation.

5.7.1 Detailed Description

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

5.7.2 Constructor & Destructor Documentation

5.7.2.1 AstNodeBreak()

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

The constructor.

Parameters

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

5.7.3 Member Function Documentation

5.7.3.1 compile()

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

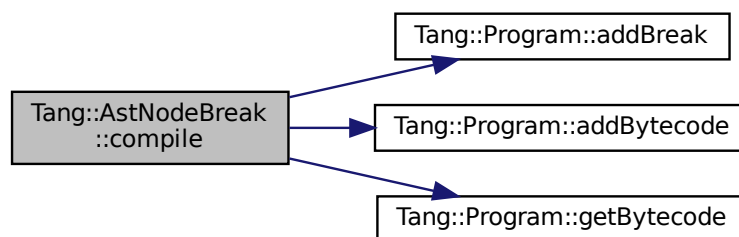
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.7.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeReturn](#), [Tang::AstNodePrint](#), [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](#).

5.7.3.3 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:

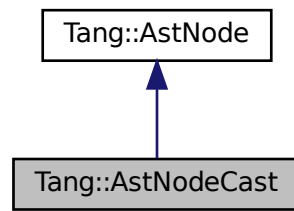
- [include/astNodeBreak.hpp](#)
- [src/astNodeBreak.cpp](#)

5.8 Tang::AstNodeCast Class Reference

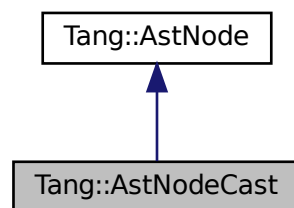
An [AstNode](#) that represents a typecast of an expression.

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

Inheritance diagram for Tang::AstNodeCast:



Collaboration diagram for Tang::AstNodeCast:



Public Types

- enum [Type](#) { [Integer](#) , [Float](#) , [Boolean](#) }
- The possible types that can be cast to.*

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) const override
Run any preprocess analysis needed before compilation.

5.8.1 Detailed Description

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

5.8.2 Member Enumeration Documentation

5.8.2.1 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 .

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]
```

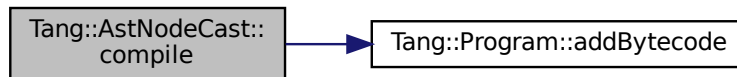
Compile the ast of the provided [Tang::Program](#).

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 ) const [override], [virtual]
```

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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:

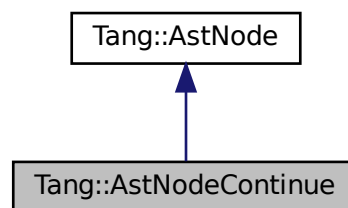
- include/[astNodeCast.hpp](#)
- src/[astNodeCast.cpp](#)

5.9 Tang::AstNodeContinue Class Reference

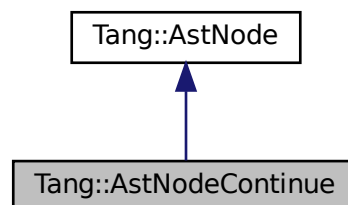
An [AstNode](#) that represents a `continue` statement.

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

Inheritance diagram for Tang::AstNodeContinue:



Collaboration diagram for Tang::AstNodeContinue:



Public 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) const
Run any preprocess analysis needed before compilation.

5.9.1 Detailed Description

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

5.9.2 Constructor & Destructor Documentation

5.9.2.1 AstNodeContinue()

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

The constructor.

Parameters

location	The location associated with the expression.
--------------------------	--

5.9.3 Member Function Documentation

5.9.3.1 compile()

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

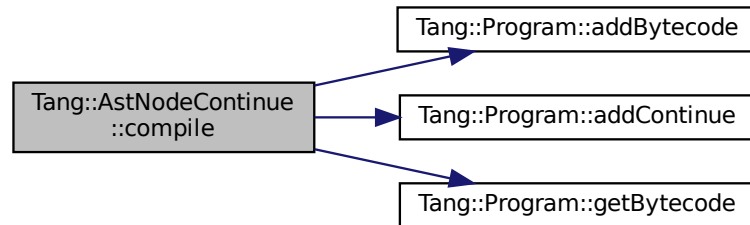
Compile the ast of the provided [Tang::Program](#).

Parameters

program	The Program which will hold the generated Bytecode.
-------------------------	---

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

Here is the call graph for this function:



5.9.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeReturn](#), [Tang::AstNodePrint](#), [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](#).

5.9.3.3 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:

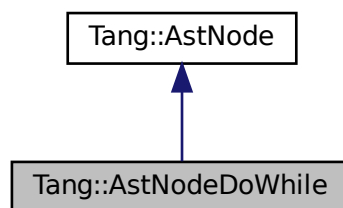
- [include/astNodeContinue.hpp](#)
- [src/astNodeContinue.cpp](#)

5.10 Tang::AstNodeDoWhile Class Reference

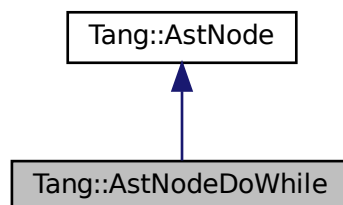
An [AstNode](#) that represents a do..while statement.

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

Inheritance diagram for Tang::AstNodeDoWhile:



Collaboration diagram for Tang::AstNodeDoWhile:



Public 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) const override

Run any preprocess analysis needed before compilation.

5.10.1 Detailed Description

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

5.10.2 Constructor & Destructor Documentation

5.10.2.1 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.

5.10.3 Member Function Documentation

5.10.3.1 compile()

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

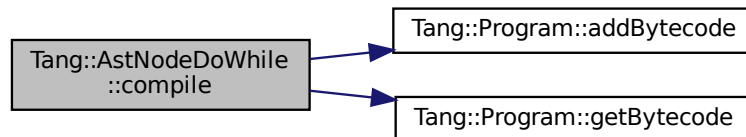
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.10.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.10.3.3 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:

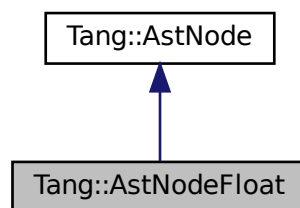
- include/[astNodeDoWhile.hpp](#)
- src/[astNodeDoWhile.cpp](#)

5.11 Tang::AstNodeFloat Class Reference

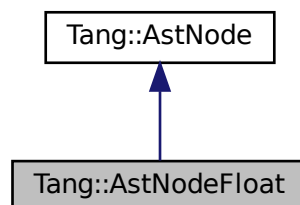
An [AstNode](#) that represents an float literal.

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

Inheritance diagram for Tang::AstNodeFloat:



Collaboration diagram for Tang::AstNodeFloat:



Public 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) const
Run any preprocess analysis needed before compilation.

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 Constructor & Destructor Documentation

5.11.2.1 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.

5.11.3 Member Function Documentation

5.11.3.1 compile()

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

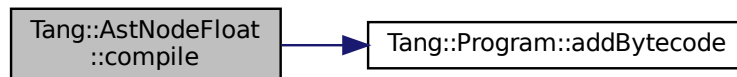
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.11.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeReturn](#), [Tang::AstNodePrint](#), [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](#).

5.11.3.3 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:

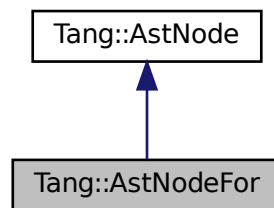
- [include/astNodeFloat.hpp](#)
- [src/astNodeFloat.cpp](#)

5.12 Tang::AstNodeFor Class Reference

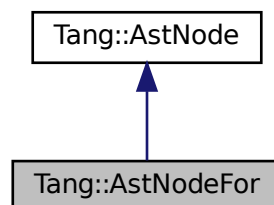
An [AstNode](#) that represents an if() statement.

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

Inheritance diagram for Tang::AstNodeFor:



Collaboration diagram for Tang::AstNodeFor:



Public 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) const override

Run any preprocess analysis needed before compilation.

5.12.1 Detailed Description

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

5.12.2 Constructor & Destructor Documentation

5.12.2.1 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.

5.12.3 Member Function Documentation

5.12.3.1 compile()

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

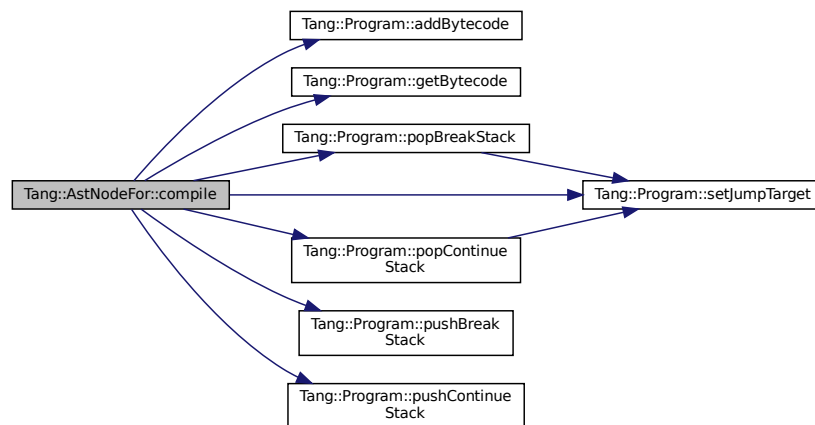
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.12.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.12.3.3 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:

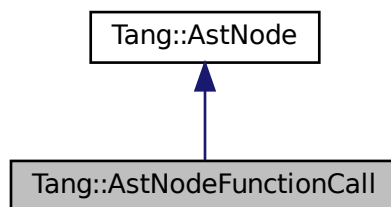
- include/[astNodeFor.hpp](#)
- src/[astNodeFor.cpp](#)

5.13 Tang::AstNodeFunctionCall Class Reference

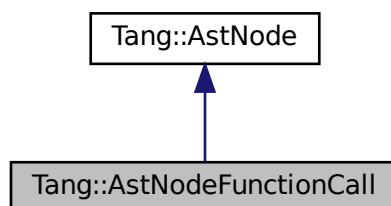
An [AstNode](#) that represents a function call.

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

Inheritance diagram for Tang::AstNodeFunctionCall:



Collaboration diagram for Tang::AstNodeFunctionCall:



Public 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) const override
Run any preprocess analysis needed before compilation.

5.13.1 Detailed Description

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

5.13.2 Constructor & Destructor Documentation

5.13.2.1 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.

5.13.3 Member Function Documentation

5.13.3.1 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:



5.13.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.13.3.3 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:

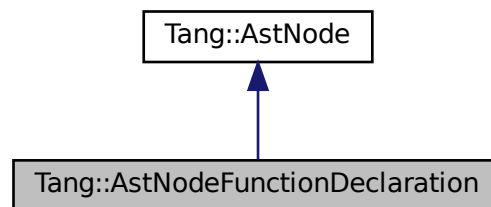
- [include/astNodeFunctionCall.hpp](#)
- [src/astNodeFunctionCall.cpp](#)

5.14 Tang::AstNodeFunctionDeclaration Class Reference

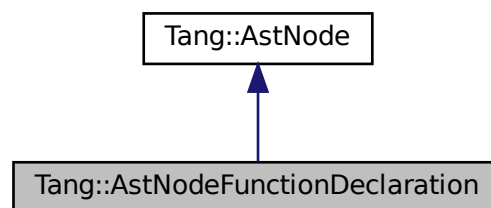
An [AstNode](#) that represents a function declaration.

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

Inheritance diagram for Tang::AstNodeFunctionDeclaration:



Collaboration diagram for Tang::AstNodeFunctionDeclaration:



Public 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) const override
Run any preprocess analysis needed before compilation.

5.14.1 Detailed Description

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

5.14.2 Constructor & Destructor Documentation

5.14.2.1 AstNodeFunctionDeclaration()

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

The constructor.

Parameters

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

5.14.3 Member Function Documentation

5.14.3.1 compile()

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

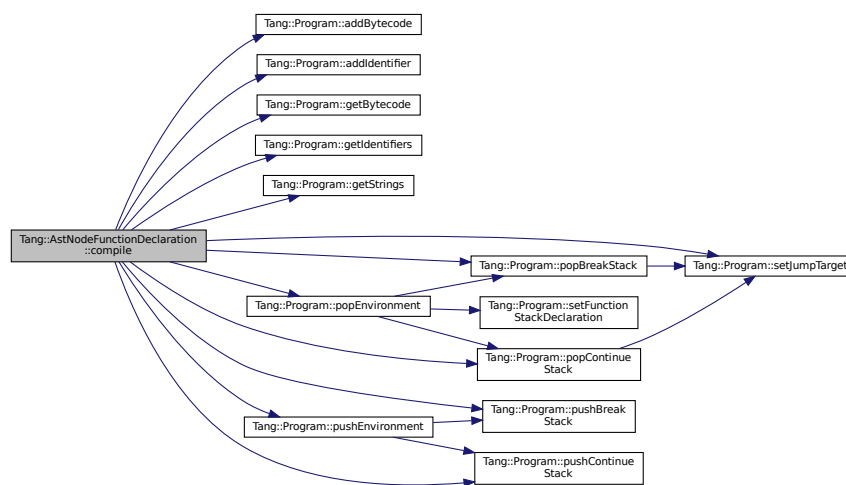
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.14.3.2 compilePreprocess()

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

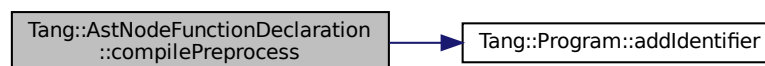
Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

Here is the call graph for this function:



5.14.3.3 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:

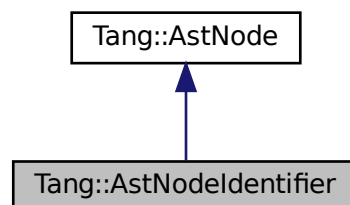
- [include/astNodeFunctionDeclaration.hpp](#)
- [src/astNodeFunctionDeclaration.cpp](#)

5.15 Tang::AstNodeIdentifier Class Reference

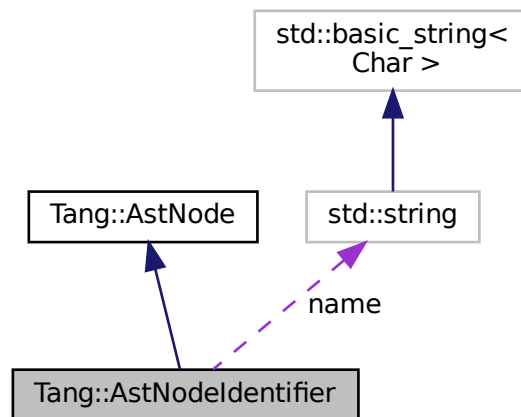
An [AstNode](#) that represents an identifier.

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

Inheritance diagram for Tang::AstNodeIdentifier:



Collaboration diagram for Tang::AstNodeIdentifier:



Public 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) const override
Run any preprocess analysis needed before compilation.

Public Attributes

- std::string [name](#)
The name of the identifier.

5.15.1 Detailed Description

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

Identifier names are represented by a string.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 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.

5.15.3 Member Function Documentation

5.15.3.1 compile()

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

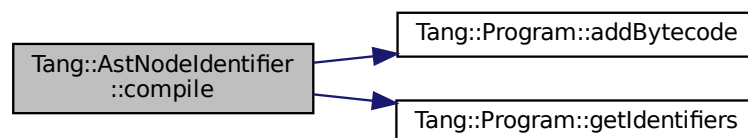
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.15.3.2 compilePreprocess()

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

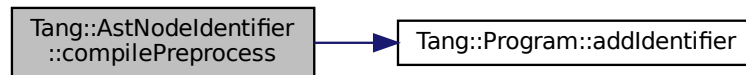
Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

Here is the call graph for this function:



5.15.3.3 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:

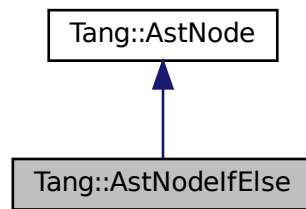
- [include/astNodeIdentifier.hpp](#)
- [src/astNodeIdentifier.cpp](#)

5.16 Tang::AstNodeIfElse Class Reference

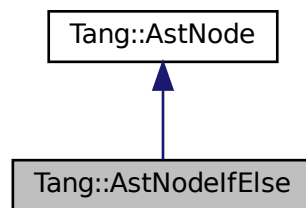
An [AstNode](#) that represents an if..else statement.

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

Inheritance diagram for Tang::AstNodeIfElse:



Collaboration diagram for Tang::AstNodeIfElse:



Public 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`) const override
Run any preprocess analysis needed before compilation.

5.16.1 Detailed Description

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

5.16.2 Constructor & Destructor Documentation

5.16.2.1 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.

5.16.2.2 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.

5.16.3 Member Function Documentation

5.16.3.1 compile()

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

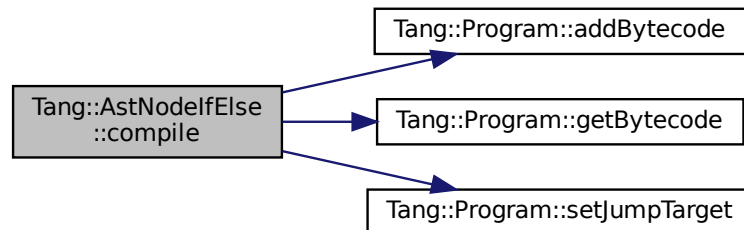
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.16.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.16.3.3 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:

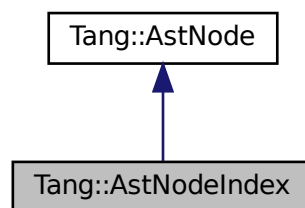
- [include/astNodeIfElse.hpp](#)
- [src/astNodeIfElse.cpp](#)

5.17 Tang::AstNodeIndex Class Reference

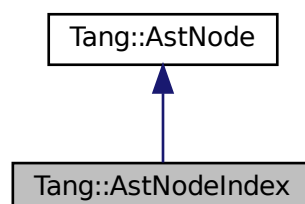
An [AstNode](#) that represents an index into a collection.

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

Inheritance diagram for Tang::AstNodeIndex:



Collaboration diagram for Tang::AstNodeIndex:



Public Member Functions

- [AstNodeIndex](#) (std::shared_ptr< [AstNode](#) > collection, std::shared_ptr< [AstNode](#) > index, [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) const override
Run any preprocess analysis needed before compilation.

5.17.1 Detailed Description

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

5.17.2 Constructor & Destructor Documentation

5.17.2.1 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.

5.17.3 Member Function Documentation

5.17.3.1 compile()

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

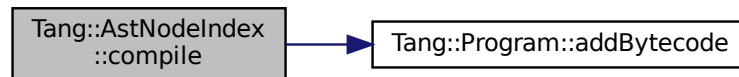
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.17.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.17.3.3 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](#).

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

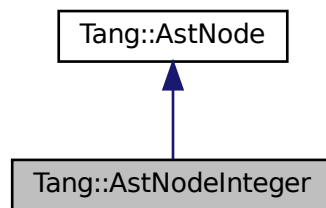
- [include/astNodeIndex.hpp](#)
- [src/astNodeIndex.cpp](#)

5.18 Tang::AstNodeInteger Class Reference

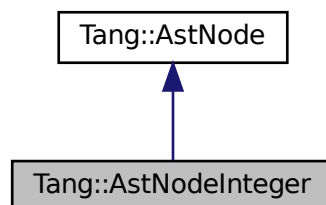
An [AstNode](#) that represents an integer literal.

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

Inheritance diagram for Tang::AstNodeInteger:



Collaboration diagram for Tang::AstNodeInteger:



Public 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) const
Run any preprocess analysis needed before compilation.

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 Constructor & Destructor Documentation

5.18.2.1 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.

5.18.3 Member Function Documentation

5.18.3.1 compile()

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

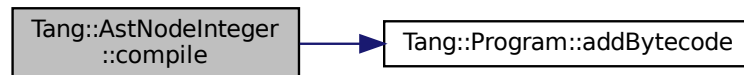
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.18.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodeTernary](#), [Tang::AstNodeString](#), [Tang::AstNodeReturn](#), [Tang::AstNodePrint](#), [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](#).

5.18.3.3 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:

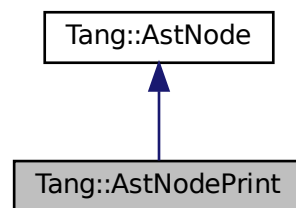
- include/[astNodeInteger.hpp](#)
- src/[astNodeInteger.cpp](#)

5.19 Tang::AstNodePrint Class Reference

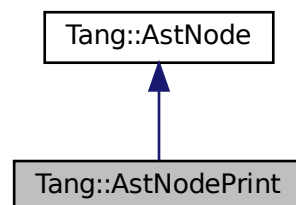
An [AstNode](#) that represents a print typeoperation.

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

Inheritance diagram for Tang::AstNodePrint:



Collaboration diagram for Tang::AstNodePrint:



Public Types

- enum [Type](#) { [Default](#) }
- The type of print() requested.*

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) const override
Run any preprocess analysis needed before compilation.

5.19.1 Detailed Description

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

5.19.2 Member Enumeration Documentation

5.19.2.1 Type

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

The type of print() requested.

Enumerator

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

5.19.3 Constructor & Destructor Documentation

5.19.3.1 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.

5.19.4 Member Function Documentation

5.19.4.1 compile()

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

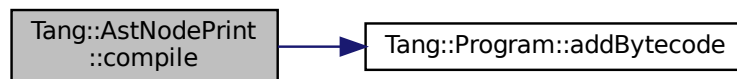
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.19.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.19.4.3 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:

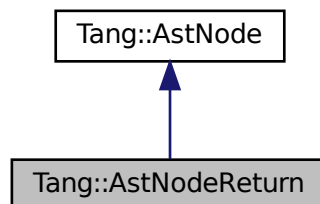
- [include/astNodePrint.hpp](#)
- [src/astNodePrint.cpp](#)

5.20 Tang::AstNodeReturn Class Reference

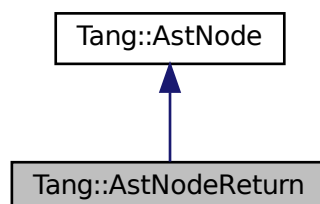
An [AstNode](#) that represents a `return` statement.

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

Inheritance diagram for Tang::AstNodeReturn:



Collaboration diagram for Tang::AstNodeReturn:



Public 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) const override
Run any preprocess analysis needed before compilation.

5.20.1 Detailed Description

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

5.20.2 Constructor & Destructor Documentation

5.20.2.1 AstNodeReturn()

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

The constructor.

Parameters

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

5.20.3 Member Function Documentation

5.20.3.1 compile()

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

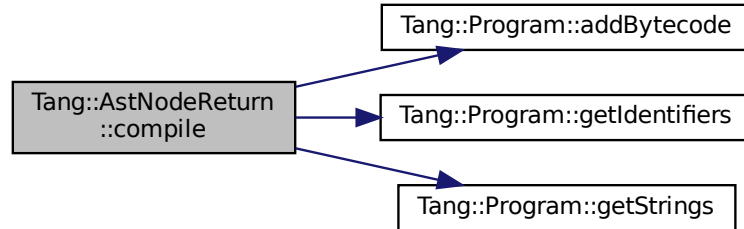
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.20.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.20.3.3 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:

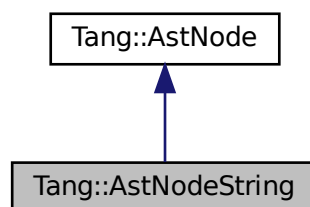
- [include/astNodeReturn.hpp](#)
- [src/astNodeReturn.cpp](#)

5.21 Tang::AstNodeString Class Reference

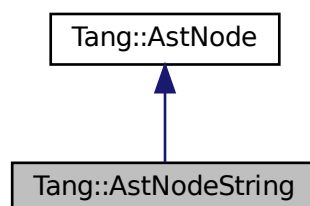
An [AstNode](#) that represents a string literal.

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

Inheritance diagram for Tang::AstNodeString:



Collaboration diagram for Tang::AstNodeString:



Public 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) const override
Run any preprocess analysis needed before compilation.
- void [compileLiteral](#) ([Tang::Program](#) &program) const
Compile the string and push it onto the stack.

5.21.1 Detailed Description

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

5.21.2 Constructor & Destructor Documentation

5.21.2.1 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.

5.21.3 Member Function Documentation

5.21.3.1 compile()

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

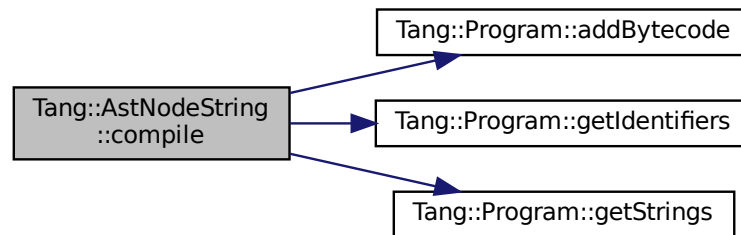
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.21.3.2 compileLiteral()

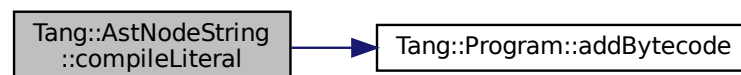
```
void AstNodeString::compileLiteral (  
    Tang::Program & program ) const
```

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:



5.21.3.3 compilePreprocess()

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

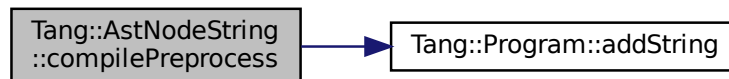
Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

Here is the call graph for this function:



5.21.3.4 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:

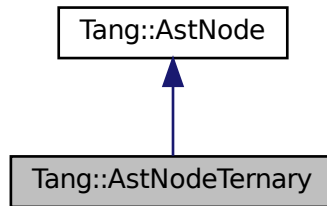
- [include/astNodeString.hpp](#)
- [src/astNodeString.cpp](#)

5.22 Tang::AstNodeTernary Class Reference

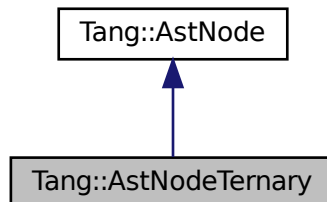
An [AstNode](#) that represents a ternary expression.

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

Inheritance diagram for Tang::AstNodeTernary:



Collaboration diagram for Tang::AstNodeTernary:



Public 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`) const override
Run any preprocess analysis needed before compilation.

5.22.1 Detailed Description

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

5.22.2 Constructor & Destructor Documentation

5.22.2.1 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.

5.22.3 Member Function Documentation

5.22.3.1 compile()

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

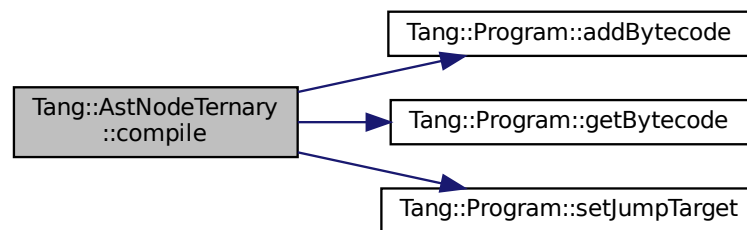
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.22.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.22.3.3 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:

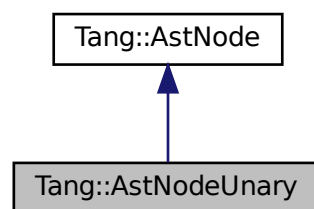
- [include/astNodeTernary.hpp](#)
- [src/astNodeTernary.cpp](#)

5.23 Tang::AstNodeUnary Class Reference

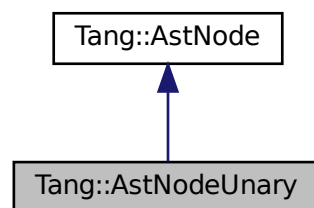
An [AstNode](#) that represents a unary negation.

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

Inheritance diagram for Tang::AstNodeUnary:



Collaboration diagram for Tang::AstNodeUnary:



Public Types

- enum [Operator](#) { [Negative](#) , [Not](#) }
- The type of operation.*

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) const override

Run any preprocess analysis needed before compilation.

5.23.1 Detailed Description

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

5.23.2 Member Enumeration Documentation

5.23.2.1 Operator

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

The type of operation.

Enumerator

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

5.23.3 Constructor & Destructor Documentation

5.23.3.1 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.

5.23.4 Member Function Documentation

5.23.4.1 compile()

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

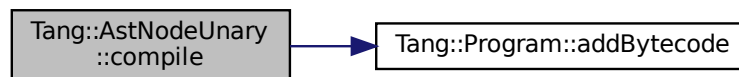
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.23.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.23.4.3 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:

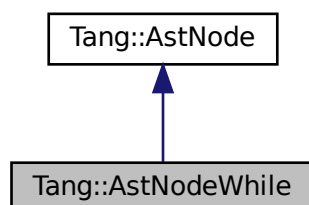
- [include/astNodeUnary.hpp](#)
- [src/astNodeUnary.cpp](#)

5.24 Tang::AstNodeWhile Class Reference

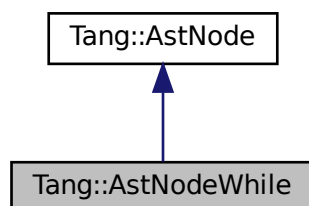
An [AstNode](#) that represents a while statement.

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

Inheritance diagram for Tang::AstNodeWhile:



Collaboration diagram for Tang::AstNodeWhile:



Public Member Functions

- [AstNodeWhile](#) (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) const override
Run any preprocess analysis needed before compilation.

5.24.1 Detailed Description

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

5.24.2 Constructor & Destructor Documentation

5.24.2.1 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.

5.24.3 Member Function Documentation

5.24.3.1 compile()

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

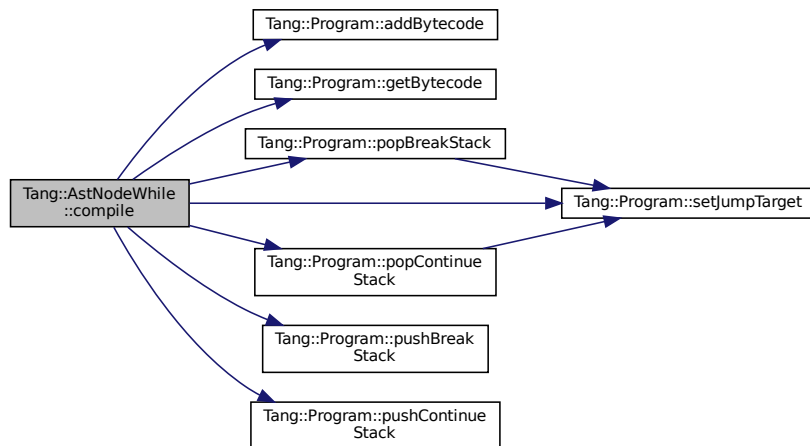
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.24.3.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

<i>program</i>	The Tang::Program that is being compiled.
----------------	---

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

5.24.3.3 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:

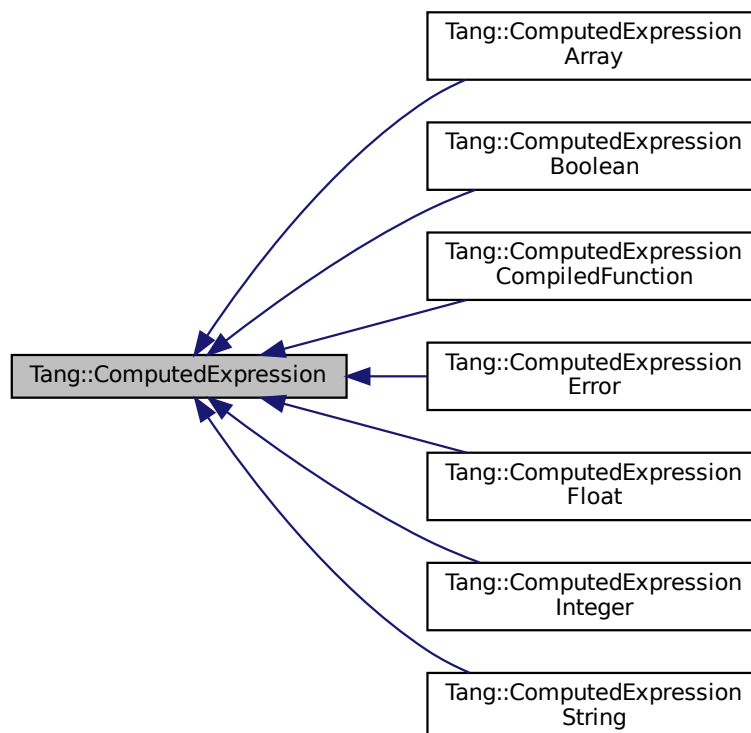
- [include/astNodeWhile.hpp](#)
- [src/astNodeWhile.cpp](#)

5.25 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 `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 __add (const GarbageCollected &rhs) const`
Compute the result of adding this value and the supplied value.
- virtual `GarbageCollected __subtract (const GarbageCollected &rhs) const`
Compute the result of subtracting this value and the supplied value.
- virtual `GarbageCollected __multiply (const GarbageCollected &rhs) const`
Compute the result of multiplying this value and the supplied value.
- virtual `GarbageCollected __divide (const GarbageCollected &rhs) const`
Compute the result of dividing this value and the supplied value.
- virtual `GarbageCollected __modulo (const GarbageCollected &rhs) const`
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __negative () const`
Compute the result of negating this value.
- virtual `GarbageCollected __not () const`
Compute the logical not of this value.
- virtual `GarbageCollected __lessThan (const GarbageCollected &rhs) const`
Compute the "less than" comparison.
- virtual `GarbageCollected __equal (const GarbageCollected &rhs) const`
Perform an equality test.
- virtual `GarbageCollected __index (const GarbageCollected &index) const`
Perform an index operation.
- virtual `GarbageCollected __integer () const`
Perform a type cast to integer.
- virtual `GarbageCollected __float () const`
Perform a type cast to float.
- virtual `GarbageCollected __boolean () const`
Perform a type cast to boolean.
- virtual `GarbageCollected __string () const`
Perform a type cast to string.

5.25.1 Detailed Description

Represents the result of a computation that has been executed.

By default, it will represent a NULL value.

5.25.2 Member Function Documentation

5.25.2.1 `__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](#).

5.25.2.2 `__boolean()`

```
GarbageCollected ComputedExpression::__boolean ( ) const [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.25.2.3 `__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](#).

5.25.2.4 `__equal()`

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

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

5.25.2.5 `__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](#).

5.25.2.6 `__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::ComputedExpressionArray](#).

5.25.2.7 __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](#).

5.25.2.8 __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](#).

5.25.2.9 __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](#).

5.25.2.10 __multiply()

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

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

Parameters

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

Returns

The result of the operation.

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

5.25.2.11 __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](#).

5.25.2.12 __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](#).

5.25.2.13 __string()

```
GarbageCollected ComputedExpression::__string ( ) const [virtual]
```

Perform a type cast to string.

Returns

The result of the the operation.

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

5.25.2.14 __subtract()

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

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

Parameters

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

Returns

The result of the operation.

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

5.25.2.15 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::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), [Tang::ComputedExpressionBoolean](#), and [Tang::ComputedExpressionArray](#).

5.25.2.16 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](#).

5.25.2.17 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](#).

5.25.2.18 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.

5.25.2.19 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](#).

5.25.2.20 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](#).

5.25.2.21 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](#).

5.25.2.22 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::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), [Tang::ComputedExpressionBoolean](#), and [Tang::ComputedExpressionArray](#).

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

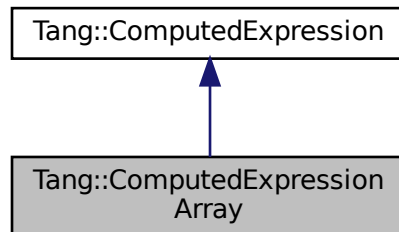
- [include/computedExpression.hpp](#)
- [src/computedExpression.cpp](#)

5.26 Tang::ComputedExpressionArray Class Reference

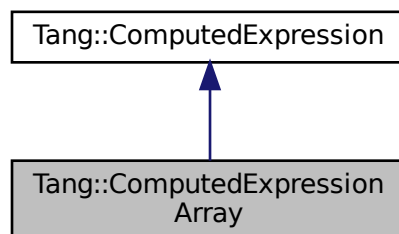
Represents an Array that is the result of a computation.

```
#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.
- [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 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
Compute the "less than" comparison.
- virtual `GarbageCollected` `__equal` (const `GarbageCollected` &rhs) const
Perform an equality test.
- virtual `GarbageCollected` `__integer` () const
Perform a type cast to integer.
- virtual `GarbageCollected` `__float` () const
Perform a type cast to float.
- virtual `GarbageCollected` `__boolean` () const
Perform a type cast to boolean.
- virtual `GarbageCollected` `__string` () const
Perform a type cast to string.

5.26.1 Detailed Description

Represents an Array that is the result of a computation.

5.26.2 Constructor & Destructor Documentation

5.26.2.1 ComputedExpressionArray()

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

Construct an Array result.

Parameters

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

5.26.3 Member Function Documentation

5.26.3.1 `__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](#).

5.26.3.2 `__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::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.26.3.3 `__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](#).

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

5.26.3.5 `__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](#).

5.26.3.6 `__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](#).

5.26.3.7 __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](#).

5.26.3.8 __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](#).

5.26.3.9 __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](#).

5.26.3.10 __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](#).

5.26.3.11 __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](#).

5.26.3.12 `__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](#).

5.26.3.13 `__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::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.26.3.14 `__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](#).

5.26.3.15 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](#).

5.26.3.16 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](#).

5.26.3.17 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](#).

5.26.3.18 `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.

5.26.3.19 `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](#).

5.26.3.20 `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](#).

5.26.3.21 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](#).

5.26.3.22 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:

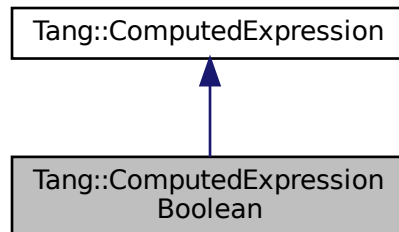
- [include/computedExpressionArray.hpp](#)
- [src/computedExpressionArray.cpp](#)

5.27 Tang::ComputedExpressionBoolean Class Reference

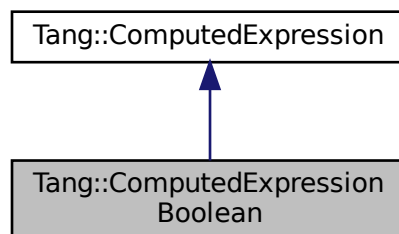
Represents an Boolean that is the result of a computation.

```
#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 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](#) `__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](#) `__index` (const [GarbageCollected](#) &index) const
Perform an index operation.
- virtual [GarbageCollected](#) `__string` () const
Perform a type cast to string.

5.27.1 Detailed Description

Represents an Boolean that is the result of a computation.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 ComputedExpressionBoolean()

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

Construct an Boolean result.

Parameters

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

5.27.3 Member Function Documentation

5.27.3.1 `__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](#).

5.27.3.2 `__boolean()`

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.27.3.3 `__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](#).

5.27.3.4 __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](#).

5.27.3.5 __float()

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.27.3.6 __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::ComputedExpressionArray](#).

5.27.3.7 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.27.3.8 __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](#).

5.27.3.9 __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](#).

5.27.3.10 __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](#).

5.27.3.11 __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](#).

5.27.3.12 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.27.3.13 __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::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.27.3.14 __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](#).

5.27.3.15 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](#).

5.27.3.16 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](#).

5.27.3.17 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](#).

5.27.3.18 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.

5.27.3.19 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](#).

5.27.3.20 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](#).

5.27.3.21 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](#).

5.27.3.22 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:

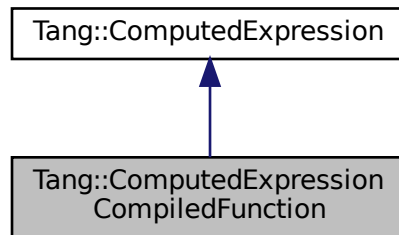
- [include/computedExpressionBoolean.hpp](#)
- [src/computedExpressionBoolean.cpp](#)

5.28 Tang::ComputedExpressionCompiledFunction Class Reference

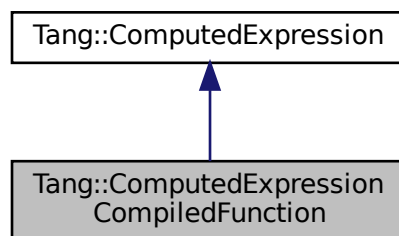
Represents a Compiled Function declared in the script.

```
#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 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
Compute the "less than" comparison.
- virtual [GarbageCollected](#) `__index` (const [GarbageCollected](#) &index) const
Perform an index 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.
- virtual [GarbageCollected](#) `__string` () const
Perform a type cast to string.

5.28.1 Detailed Description

Represents a Compiled Function declared in the script.

5.28.2 Constructor & Destructor Documentation

5.28.2.1 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.

5.28.3 Member Function Documentation

5.28.3.1 __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](#).

5.28.3.2 __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::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.28.3.3 `__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](#).

5.28.3.4 `__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](#).

5.28.3.5 `__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](#).

5.28.3.6 __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::ComputedExpressionArray](#).

5.28.3.7 __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](#).

5.28.3.8 __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](#).

5.28.3.9 __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](#).

5.28.3.10 __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](#).

5.28.3.11 __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](#).

5.28.3.12 __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](#).

5.28.3.13 __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::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.28.3.14 __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](#).

5.28.3.15 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](#).

5.28.3.16 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](#).

5.28.3.17 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](#).

5.28.3.18 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.

5.28.3.19 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](#).

5.28.3.20 `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](#).

5.28.3.21 `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](#).

5.28.3.22 `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:

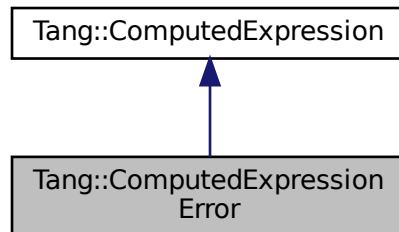
- [include/computedExpressionCompiledFunction.hpp](#)
- [src/computedExpressionCompiledFunction.cpp](#)

5.29 Tang::ComputedExpressionError Class Reference

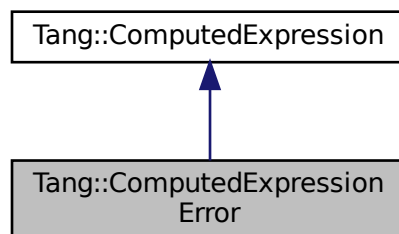
Represents a Runtime [Error](#).

```
#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
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.
- 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 __index` (const `GarbageCollected` &index) const
Perform an index operation.

5.29.1 Detailed Description

Represents a Runtime `Error`.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 ComputedExpressionError()

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

Construct a Runtime `Error`.

Parameters

<i>error</i>	The Tang::Error object.
--------------	---

5.29.3 Member Function Documentation

5.29.3.1 `__add()`

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

5.29.3.2 `__boolean()`

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.29.3.3 `__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](#).

5.29.3.4 `__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](#).

5.29.3.5 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.29.3.6 `__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::ComputedExpressionArray](#).

5.29.3.7 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.29.3.8 __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](#).

5.29.3.9 __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](#).

5.29.3.10 `__multiply()`

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

5.29.3.11 `__negative()`

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.29.3.12 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.29.3.13 __string()

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

Perform a type cast to string.

Returns

The result of the the operation.

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

5.29.3.14 __subtract()

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

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

Parameters

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

Returns

The result of the operation.

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

5.29.3.15 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](#).

5.29.3.16 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](#).

5.29.3.17 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](#).

5.29.3.18 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.

5.29.3.19 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](#).

5.29.3.20 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](#).

5.29.3.21 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](#).

5.29.3.22 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:

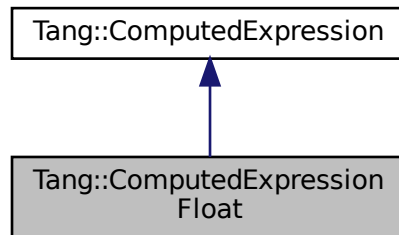
- [include/computedExpressionError.hpp](#)
- [src/computedExpressionError.cpp](#)

5.30 Tang::ComputedExpressionFloat Class Reference

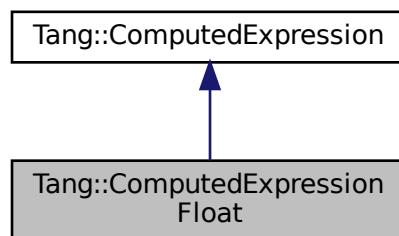
Represents a Float that is the result of a computation.

```
#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.
- 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 __modulo` (const `GarbageCollected` &rhs) const
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __index` (const `GarbageCollected` &index) const
Perform an index operation.

Friends

- class `ComputedExpressionInteger`

5.30.1 Detailed Description

Represents a Float that is the result of a computation.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 ComputedExpressionFloat()

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

Construct a Float result.

Parameters

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

5.30.3 Member Function Documentation

5.30.3.1 __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](#).

5.30.3.2 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.30.3.3 __divide()

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

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

Parameters

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

Returns

The result of the operation.

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

5.30.3.4 `__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](#).

5.30.3.5 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.30.3.6 `__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::ComputedExpressionArray](#).

5.30.3.7 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.30.3.8 __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](#).

5.30.3.9 __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](#).

5.30.3.10 __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](#).

5.30.3.11 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.30.3.12 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.30.3.13 __string()

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

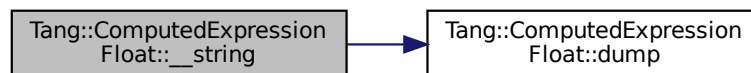
Perform a type cast to string.

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.30.3.14 __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](#).

5.30.3.15 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](#).

5.30.3.16 is_equal() [1/6]

```
bool ComputedExpressionFloat::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](#).

5.30.3.17 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](#).

5.30.3.18 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.

5.30.3.19 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](#).

5.30.3.20 `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](#).

5.30.3.21 `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](#).

5.30.3.22 `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:

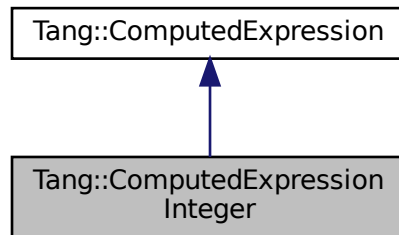
- include/computedExpressionFloat.hpp
- src/computedExpressionFloat.cpp

5.31 Tang::ComputedExpressionInteger Class Reference

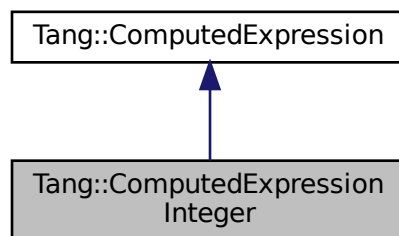
Represents an Integer that is the result of a computation.

```
#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.
- 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 __index` (const `GarbageCollected` &index) const
Perform an index operation.

Friends

- class `ComputedExpressionFloat`
- class `ComputedExpressionArray`

5.31.1 Detailed Description

Represents an Integer that is the result of a computation.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 ComputedExpressionInteger()

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

Construct an Integer result.

Parameters

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

5.31.3 Member Function Documentation

5.31.3.1 __add()

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

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

Parameters

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

Returns

The result of the operation.

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

5.31.3.2 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.31.3.3 __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](#).

5.31.3.4 `__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](#).

5.31.3.5 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.31.3.6 `__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::ComputedExpressionArray](#).

5.31.3.7 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.31.3.8 __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](#).

5.31.3.9 __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](#).

5.31.3.10 `__multiply()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.31.3.11 `__negative()`

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.31.3.12 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.31.3.13 __string()

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

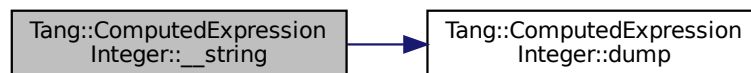
Perform a type cast to string.

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.31.3.14 __subtract()**

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

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

Parameters

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

Returns

The result of the operation.

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

5.31.3.15 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](#).

5.31.3.16 is_equal() [1/6]

```
bool ComputedExpressionInteger::is_equal (
    const bool & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

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

Returns

True if equal, false if not.

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

5.31.3.17 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](#).

5.31.3.18 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.

5.31.3.19 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](#).

5.31.3.20 `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

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

Returns

True if equal, false if not.

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

5.31.3.21 `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

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

Returns

True if equal, false if not.

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

5.31.3.22 `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:

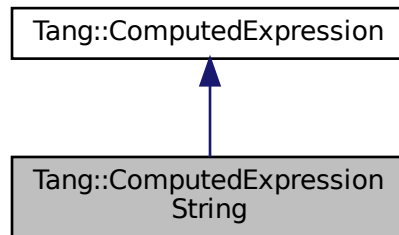
- [include/computedExpressionInteger.hpp](#)
- [src/computedExpressionInteger.cpp](#)

5.32 Tang::ComputedExpressionString Class Reference

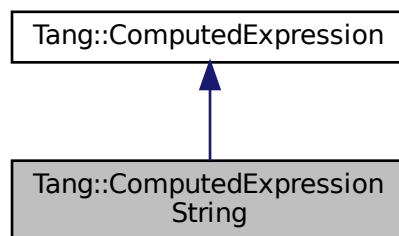
Represents a String that is the result of a computation.

```
#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.
- [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 __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.
- 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 __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 __index` (const `GarbageCollected` &index) const
Perform an index operation.
- virtual `GarbageCollected __integer` () const
Perform a type cast to integer.
- virtual `GarbageCollected __float` () const
Perform a type cast to float.

5.32.1 Detailed Description

Represents a String that is the result of a computation.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 ComputedExpressionString()

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

Construct a String result.

Parameters

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

5.32.3 Member Function Documentation

5.32.3.1 __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](#).

5.32.3.2 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.32.3.3 __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](#).

5.32.3.4 `__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](#).

5.32.3.5 `__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](#).

5.32.3.6 `__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::ComputedExpressionArray](#).

5.32.3.7 __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](#).

5.32.3.8 __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](#).

5.32.3.9 `__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](#).

5.32.3.10 __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](#).

5.32.3.11 __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](#).

5.32.3.12 `__not()`

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.32.3.13 `__string()`

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

Perform a type cast to string.

Returns

The result of the the operation.

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

5.32.3.14 `__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](#).

5.32.3.15 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](#).

5.32.3.16 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](#).

5.32.3.17 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](#).

5.32.3.18 `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.

5.32.3.19 `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](#).

5.32.3.20 `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](#).

5.32.3.21 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](#).

5.32.3.22 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:

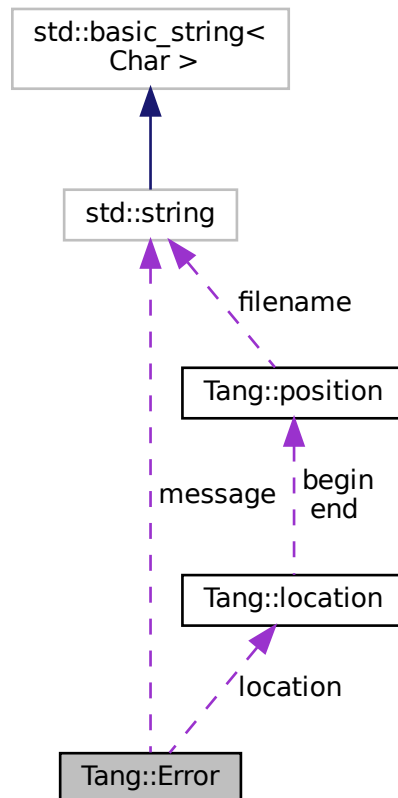
- [include/computedExpressionString.hpp](#)
- [src/computedExpressionString.cpp](#)

5.33 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.

5.33.1 Detailed Description

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

5.33.2 Constructor & Destructor Documentation

5.33.2.1 Error() [1/2]

```
Tang::Error::Error (
    std::string message ) [inline]
```

Creates an error message using the supplied error string and location.

Parameters

<i>message</i>	The error message as a string.
----------------	--------------------------------

5.33.2.2 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.

5.33.3 Friends And Related Function Documentation

5.33.3.1 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:

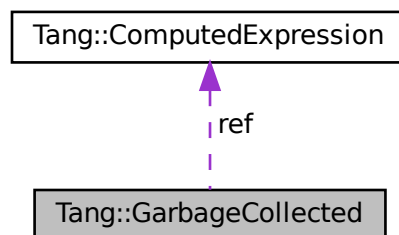
- include/[error.hpp](#)
- src/[error.cpp](#)

5.34 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.
- [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 \geq between two [GarbageCollected](#) values.*
 - [GarbageCollected operator==](#) (const [GarbageCollected](#) &rhs) const
Perform a $=$ between two [GarbageCollected](#) values.
 - [GarbageCollected operator!=](#) (const [GarbageCollected](#) &rhs) const
Perform a \neq 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.

5.34.1 Detailed Description

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

Uses the [SingletonObjectPool](#) to created and recycle object memory. The container is not thread-safe.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 [GarbageCollected\(\)](#) [1/3]

```
Tang::GarbageCollected::GarbageCollected (
    const GarbageCollected & other ) [inline]
```

Copy Constructor.

Parameters

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

5.34.2.2 GarbageCollected() [2/3]

```
Tang::GarbageCollected::GarbageCollected (
    GarbageCollected && other ) [inline]
```

Move Constructor.

Parameters

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

5.34.2.3 ~GarbageCollected()

```
Tang::GarbageCollected::~~GarbageCollected ( ) [inline]
```

Destructor.

Clean up the tracked object, if appropriate.

5.34.2.4 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.
-----------------	---

5.34.3 Member Function Documentation**5.34.3.1 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:



5.34.3.2 operator"!")()

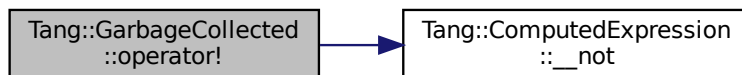
```
GarbageCollected GarbageCollected::operator! ( ) const
```

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

Returns

The result of the operation.

Here is the call graph for this function:



5.34.3.3 operator"!=()"

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

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

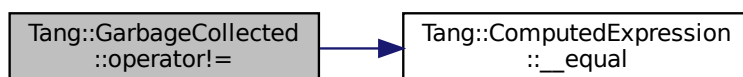
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.34.3.4 operator%()

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

Perform a modulo between two `GarbageCollected` values.

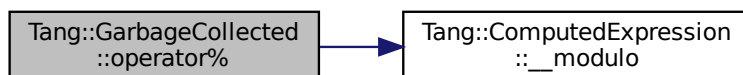
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.34.3.5 operator*() [1/2]

```
ComputedExpression& Tang::GarbageCollected::operator* ( ) const [inline]
```

Access the tracked object.

Returns

A reference to the tracked object.

5.34.3.6 operator*() [2/2]

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

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

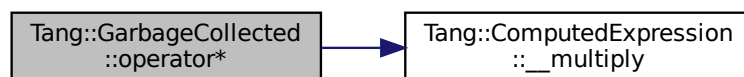
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:

**5.34.3.7 operator+()**

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

Perform an addition between two [GarbageCollected](#) values.

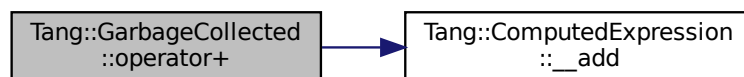
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.34.3.8 operator-() [1/2]

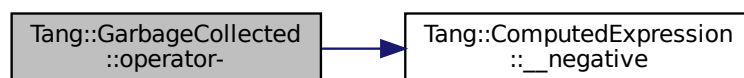
```
GarbageCollected GarbageCollected::operator- ( ) const
```

Perform a negation on the `GarbageCollected` value.

Returns

The result of the operation.

Here is the call graph for this function:



5.34.3.9 operator-() [2/2]

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

Perform a subtraction between two `GarbageCollected` values.

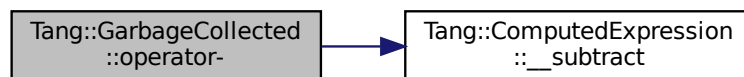
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:

**5.34.3.10 operator->()**

```
ComputedExpression* Tang::GarbageCollected::operator-> ( ) const [inline]
```

Access the tracked object as a pointer.

Returns

A pointer to the tracked object.

5.34.3.11 operator/()

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

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

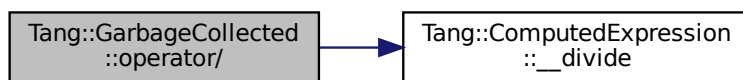
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.34.3.12 `operator<()`

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

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

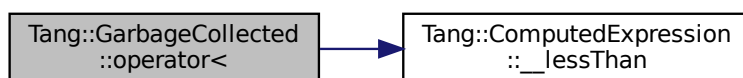
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.34.3.13 `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.

5.34.3.14 **operator=()** [1/2]

```
GarbageCollected& Tang::GarbageCollected::operator= (
    const GarbageCollected & other ) [inline]
```

Copy Assignment.

Parameters

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

Here is the call graph for this function:

5.34.3.15 **operator=()** [2/2]

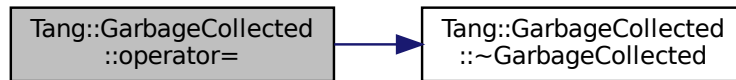
```
GarbageCollected& Tang::GarbageCollected::operator= (
    GarbageCollected && other ) [inline]
```

Move Assignment.

Parameters

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

Here is the call graph for this function:



5.34.3.16 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.

5.34.3.17 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.

5.34.3.18 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.

5.34.3.19 operator==() [4/8]

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

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

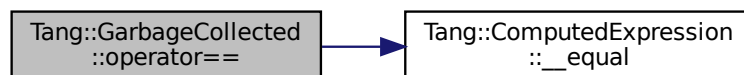
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:

**5.34.3.20 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.

5.34.3.21 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.

5.34.3.22 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.

5.34.3.23 operator==() [8/8]

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

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

Parameters

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

Returns

True if they are equal, false otherwise.

5.34.3.24 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.

5.34.3.25 operator>=()

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

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

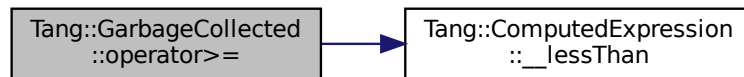
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.34.4 Friends And Related Function Documentation

5.34.4.1 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:

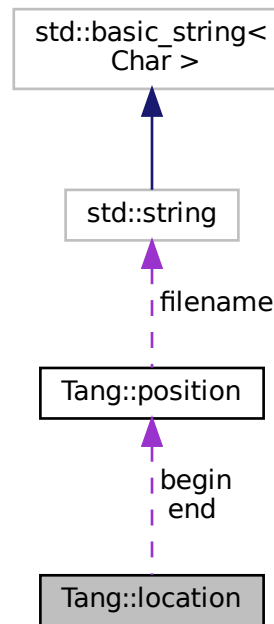
- [include/garbageCollected.hpp](#)
- [src/garbageCollected.cpp](#)

5.35 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.

5.35.1 Detailed Description

Two points in a source file.

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

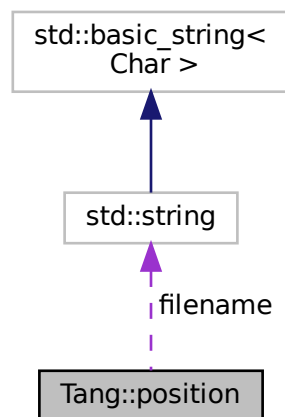
- [build/generated/location.hh](#)

5.36 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.

5.36.1 Detailed Description

A point in a source file.

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

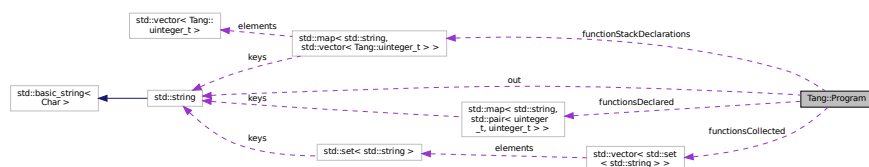
- `build/generated/location.hh`

5.37 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)
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 [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.

5.37.1 Detailed Description

Represents a compiled script or template that may be executed.

5.37.2 Member Enumeration Documentation

5.37.2.1 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.

5.37.3 Constructor & Destructor Documentation

5.37.3.1 Program()

```
Program::Program (
    std::string code,
    Program::CodeType codeType )
```

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 <code>Script</code> or <code>Template</code> .

5.37.4 Member Function Documentation

5.37.4.1 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.
-----------------	---

5.37.4.2 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.

5.37.4.3 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.
-----------------	--

5.37.4.4 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.

5.37.4.5 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.

5.37.4.6 dumpBytecode()

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

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

Returns

A string containing the Opcode representation.

5.37.4.7 execute()

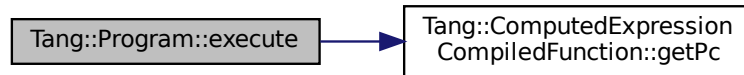
```
Program & Program::execute ( )
```

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

Returns

The current [Program](#) object.

Here is the call graph for this function:

**5.37.4.8 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.

5.37.4.9 getBytecode()

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

Get the Bytecode vector.

Returns

The Bytecode vector.

5.37.4.10 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.

5.37.4.11 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.

5.37.4.12 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.

5.37.4.13 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.

5.37.4.14 popBreakStack()

```
void Program::popBreakStack (
    size_t target )
```

For all `continue` bytecode locations collected by `Tang::addContinue`, set the target pc to `target`.

Parameters

<i>target</i>	The target bytecode offset that the <code>continue</code> should jump to.
---------------	---

Here is the call graph for this function:



5.37.4.15 popContinueStack()

```
void Program::popContinueStack (
    size_t target )
```

For all `continue` bytecode locations collected by `Tang::addContinue`, set the target pc to `target`.

Parameters

<i>target</i>	The target bytecode offset that the <code>continue</code> should jump to.
---------------	---

Here is the call graph for this function:



5.37.4.16 pushEnvironment()

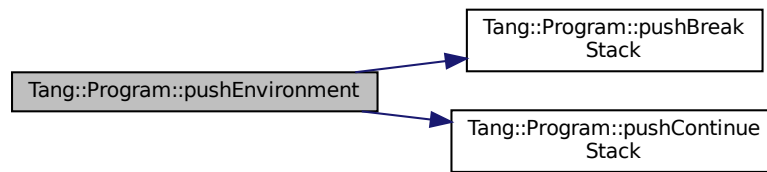
```
void Program::pushEnvironment (
    const std::shared_ptr< AstNode > & ast )
```

Create a new compile/execute environment stack entry.

Parameters

<i>ast</i>	The ast node from which this new environment will be formed.
------------	--

Here is the call graph for this function:



5.37.4.17 setFunctionStackDeclaration()

```
bool Program::setFunctionStackDeclaration (
    size_t opcodeAddress,
    unsigned_t argc,
    unsigned_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.

5.37.4.18 setJumpTarget()

```
bool Program::setJumpTarget (
    size_t opcodeAddress,
    Tang::unsigned_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.

5.37.5 Member Data Documentation

5.37.5.1 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](#)

5.38 Tang::SingletonObjectPool< T > Class Template Reference

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

```
#include <singletonObjectPool.hpp>
```

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.

5.38.1 Detailed Description

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

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

5.38.2 Member Function Documentation

5.38.2.1 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.

5.38.2.2 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.

5.38.2.3 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.
------------	---------------------------------

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

- [include/singletonObjectPool.hpp](#)

5.39 Tang::TangBase Class Reference

The base class for the Tang programming language.

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

Public Member Functions

- [TangBase](#) ()
The constructor.
- [Program compileScript](#) (std::string script)
Compile the provided source code as a script and return a [Program](#).

5.39.1 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).

5.39.2 Constructor & Destructor Documentation

5.39.2.1 TangBase()

```
TangBase::TangBase ( )
```

The constructor.

Isn't it glorious.

5.39.3 Member Function Documentation

5.39.3.1 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.

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

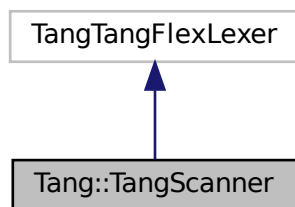
- [include/tangBase.hpp](#)
- [src/tangBase.cpp](#)

5.40 Tang::TangScanner Class Reference

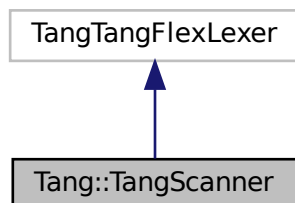
The Flex lexer class for the main Tang language.

```
#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.

5.40.1 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.

5.40.2 Constructor & Destructor Documentation

5.40.2.1 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. It's 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)

5.40.3 Member Function Documentation

5.40.3.1 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](#)

Chapter 6

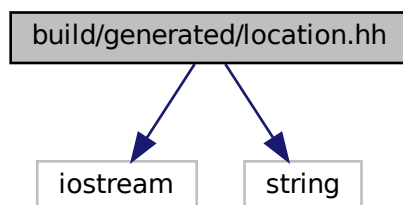
File Documentation

6.1 build/generated/location.hh File Reference

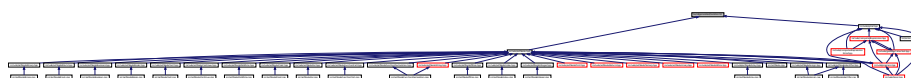
Define the Tang ::location class.

```
#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.

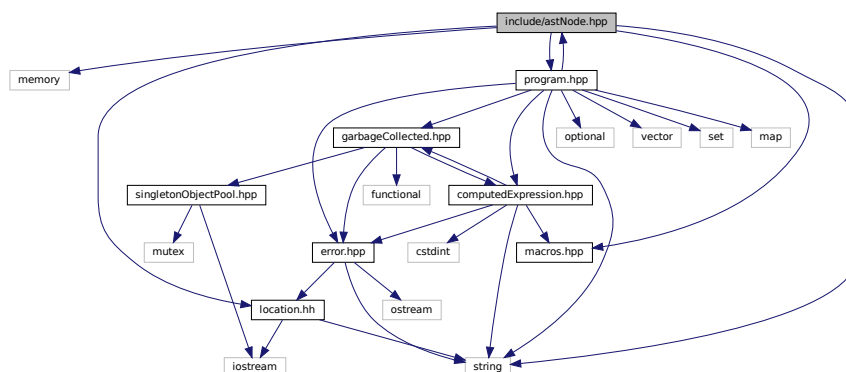
Parameters

<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).

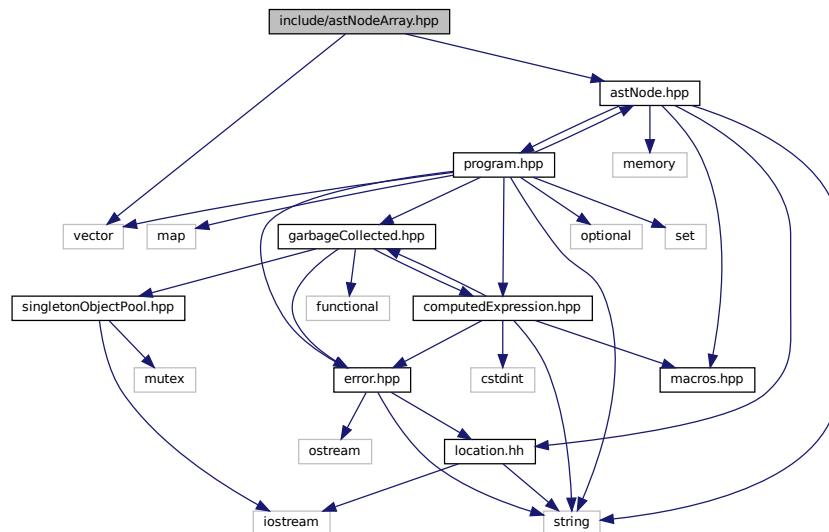
6.2.1 Detailed Description

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

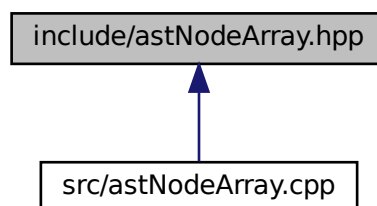
6.3 include/astNodeArray.hpp File Reference

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.

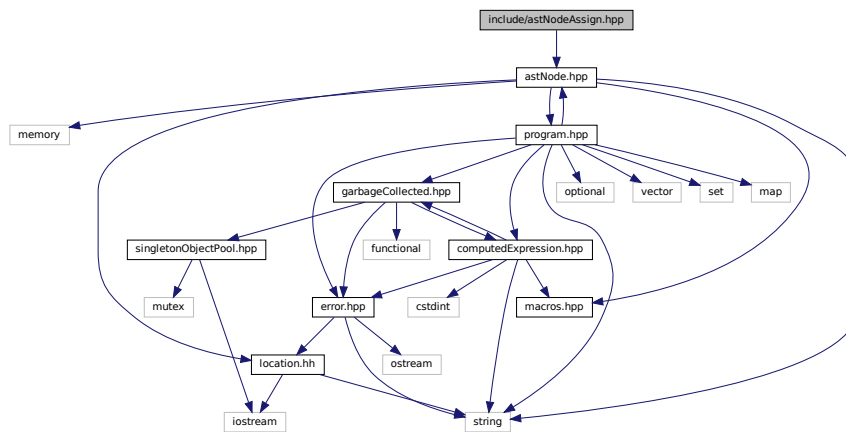
6.3.1 Detailed Description

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

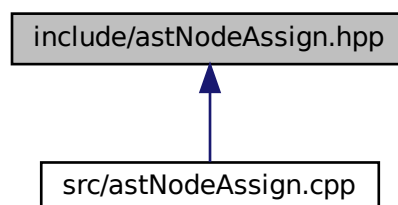
6.4 include/astNodeAssign.hpp File Reference

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.

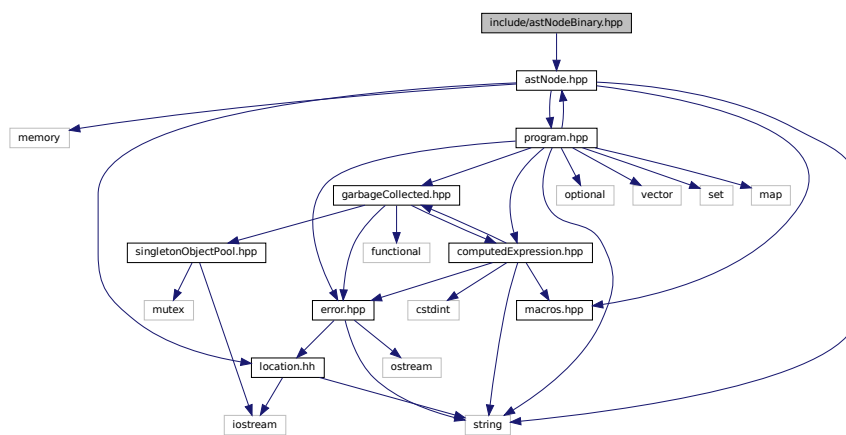
6.4.1 Detailed Description

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

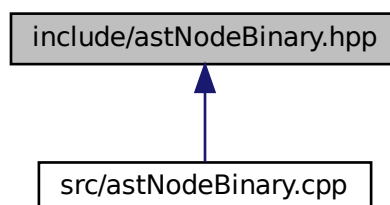
6.5 include/astNodeBinary.hpp File Reference

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

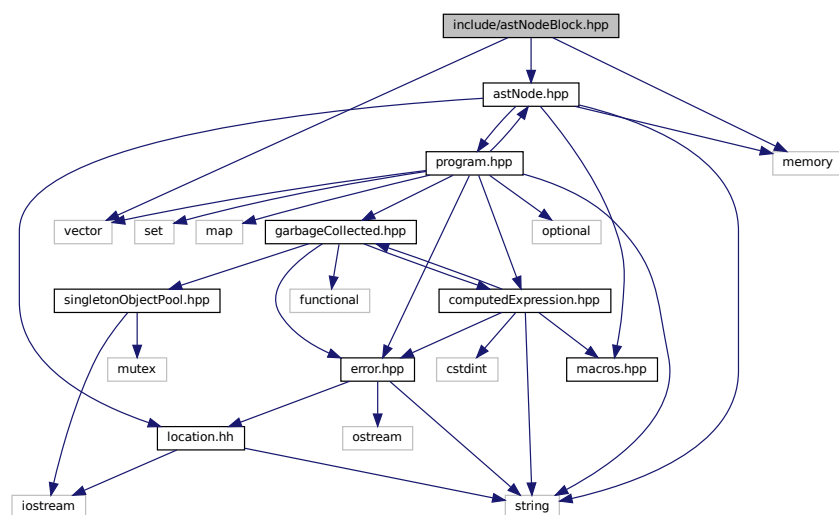
Declare the [Tang::AstNodeBinary](#) class.

6.6 include/astNodeBlock.hpp File Reference

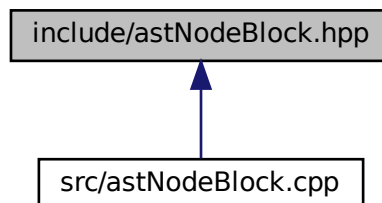
Declare the [Tang::AstNodeBlock](#) class.

```
#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

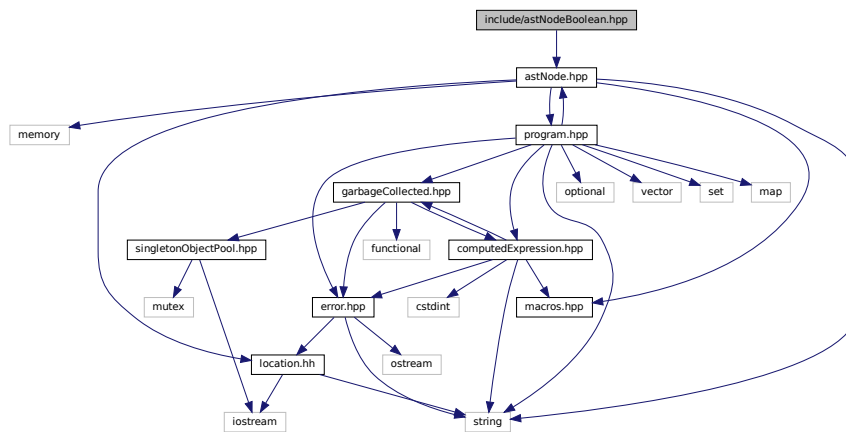
Declare the [Tang::AstNodeBlock](#) class.

6.7 include/astNodeBoolean.hpp File Reference

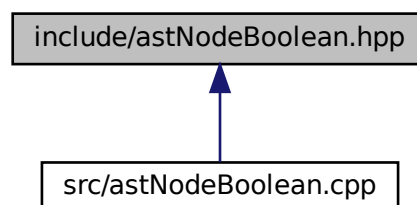
Declare the [Tang::AstNodeBoolean](#) class.

```
#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.

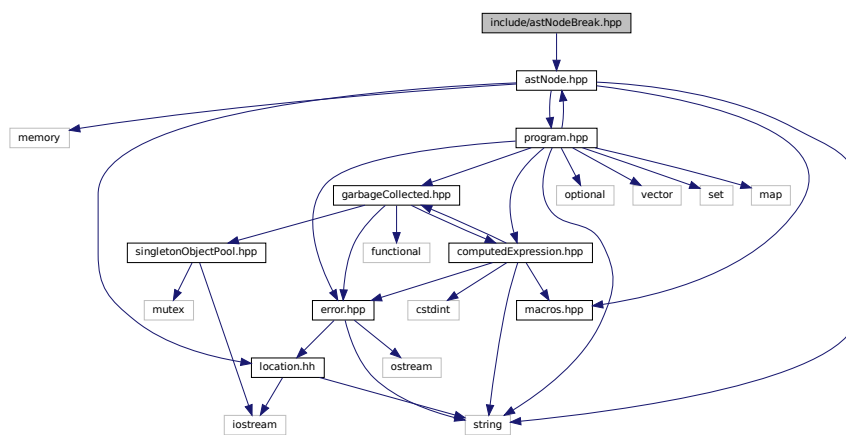
6.7.1 Detailed Description

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

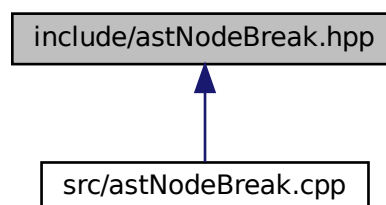
6.8 include/astNodeBreak.hpp File Reference

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.

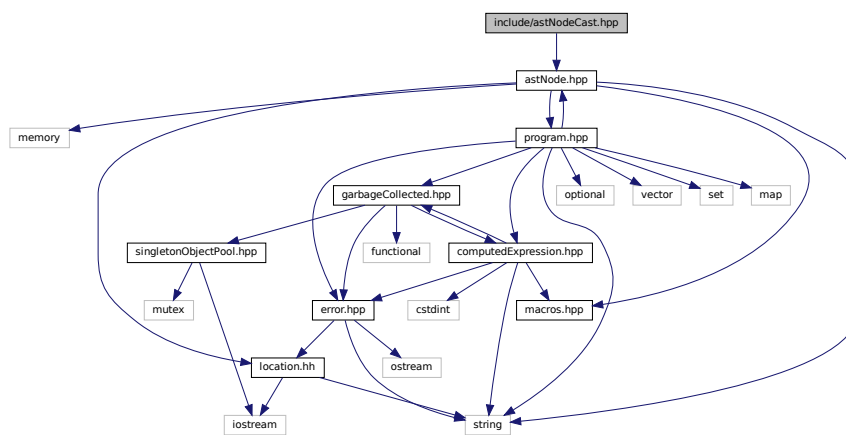
6.8.1 Detailed Description

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

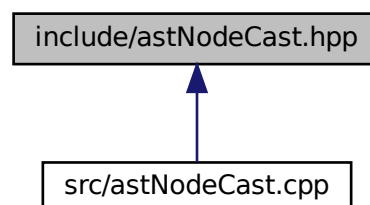
6.9 include/astNodeCast.hpp File Reference

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

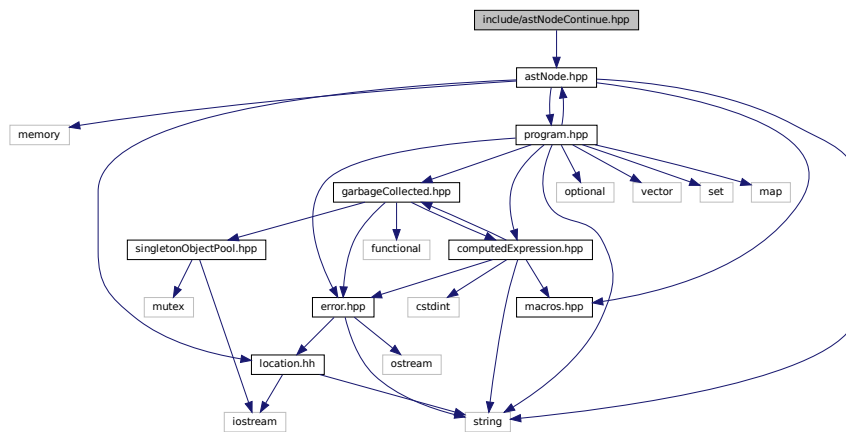
Declare the [Tang::AstNodeCast](#) class.

6.10 include/astNodeContinue.hpp File Reference

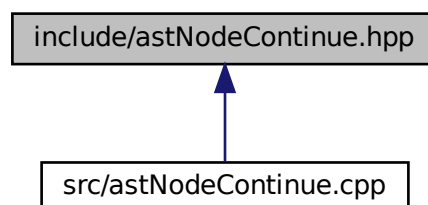
Declare the [Tang::AstNodeContinue](#) class.

```
#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

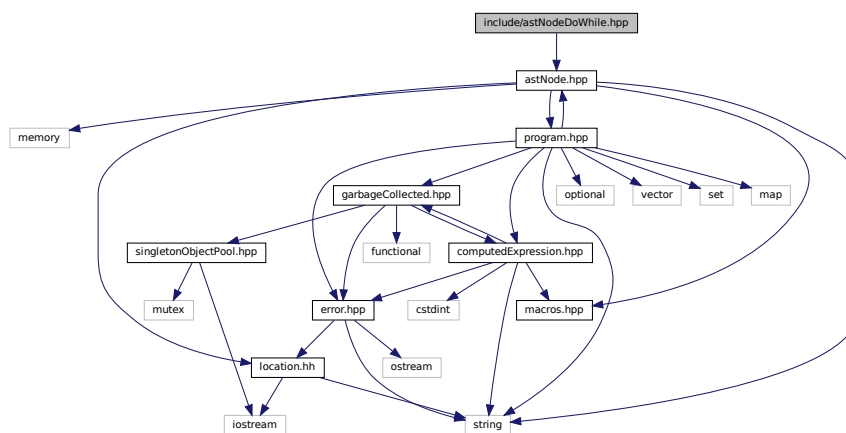
Declare the [Tang::AstNodeContinue](#) class.

6.11 include/astNodeDoWhile.hpp File Reference

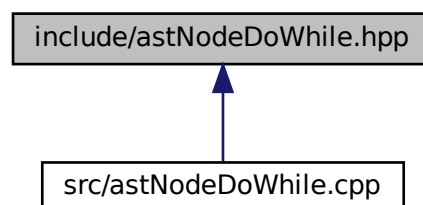
Declare the [Tang::AstNodeDoWhile](#) class.

```
#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.

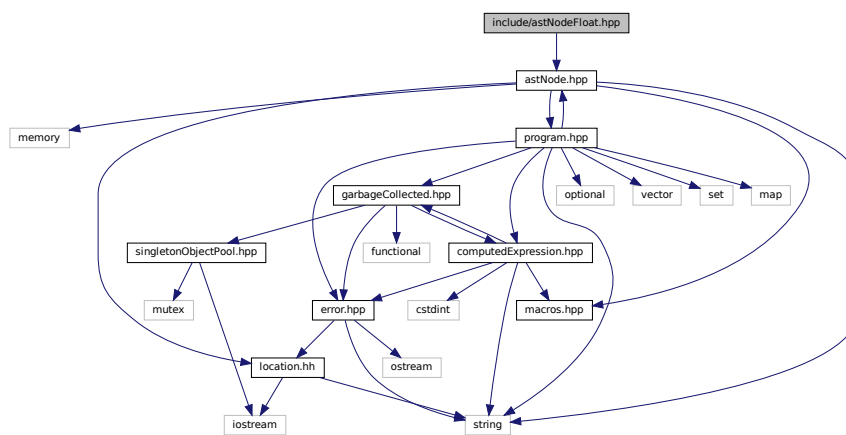
6.11.1 Detailed Description

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

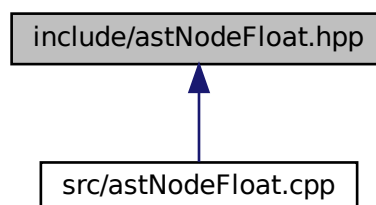
6.12 include/astNodeFloat.hpp File Reference

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.

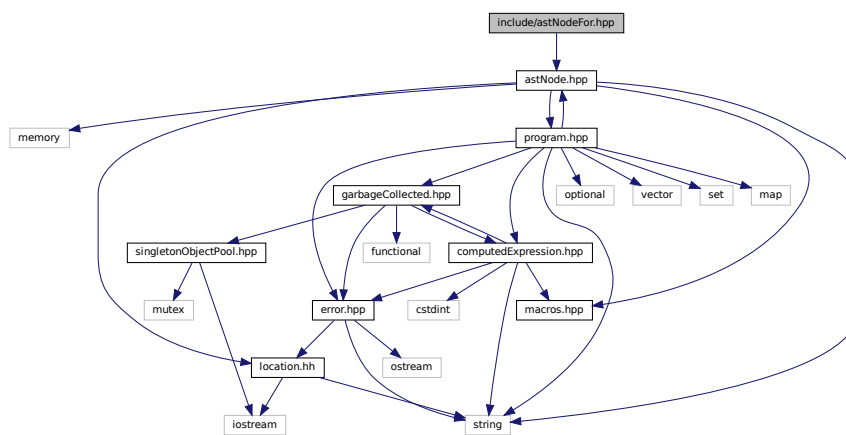
6.12.1 Detailed Description

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

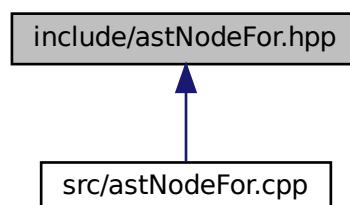
6.13 include/astNodeFor.hpp File Reference

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

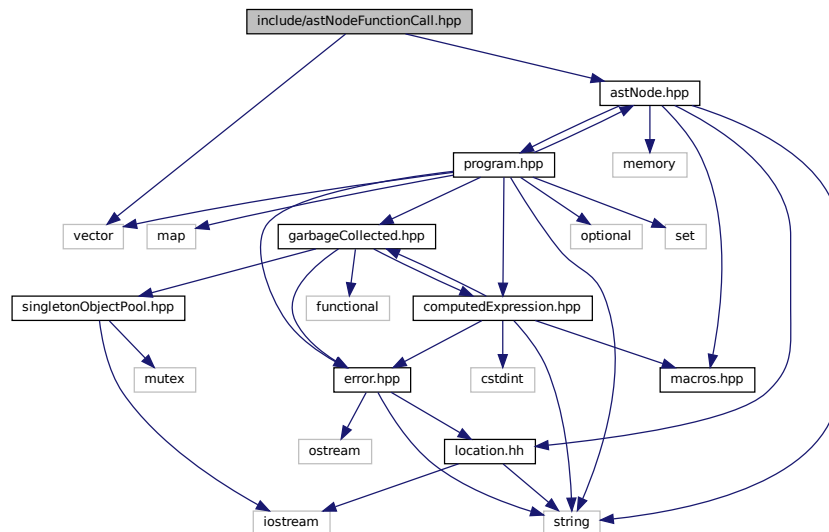
Declare the [Tang::AstNodeFor](#) class.

6.14 include/astNodeFunctionCall.hpp File Reference

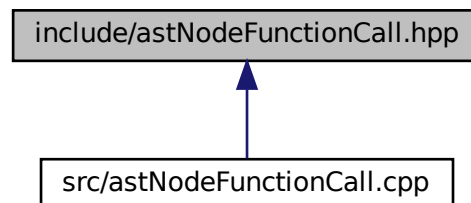
Declare the [Tang::AstNodeFunctionCall](#) class.

```
#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.

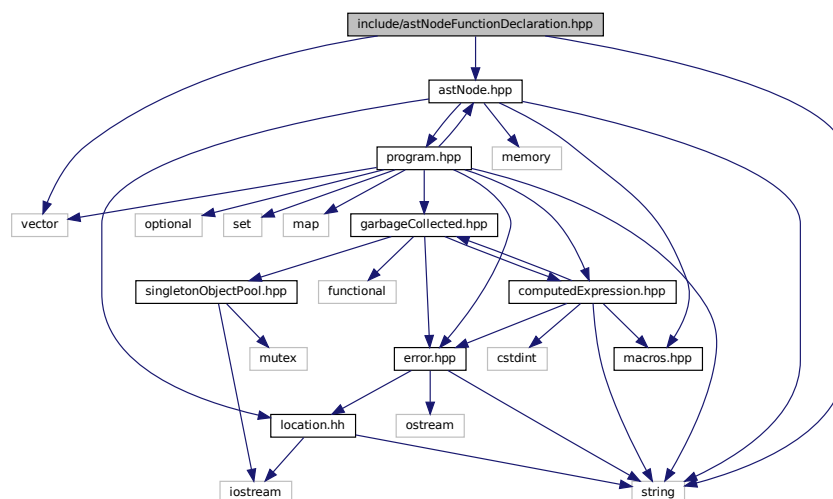
6.14.1 Detailed Description

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

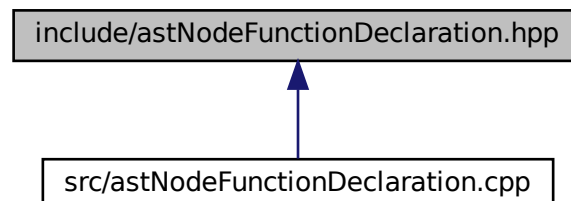
6.15 include/astNodeFunctionDeclaration.hpp File Reference

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.

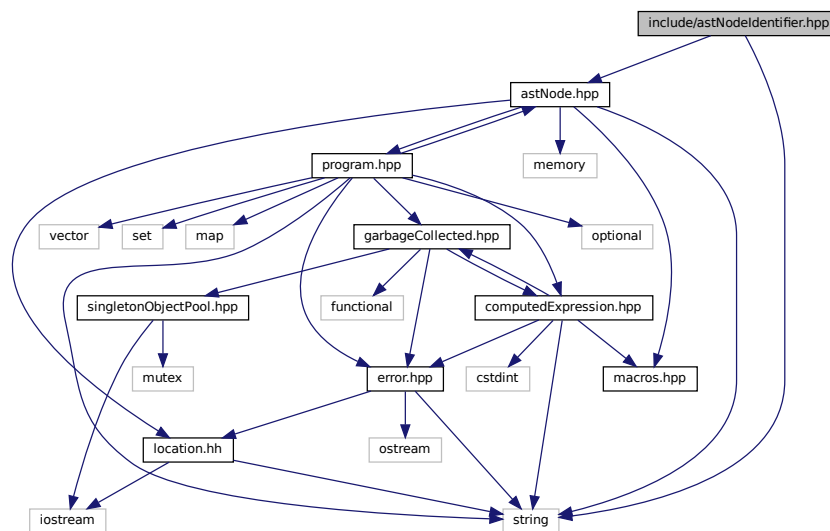
6.15.1 Detailed Description

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

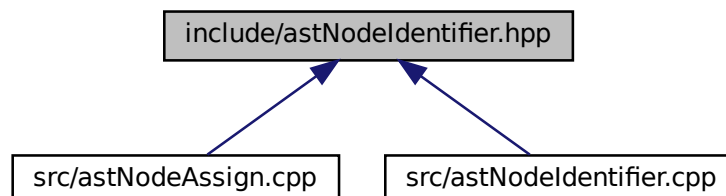
6.16 include/astNodeIdentifier.hpp File Reference

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

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



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



Classes

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

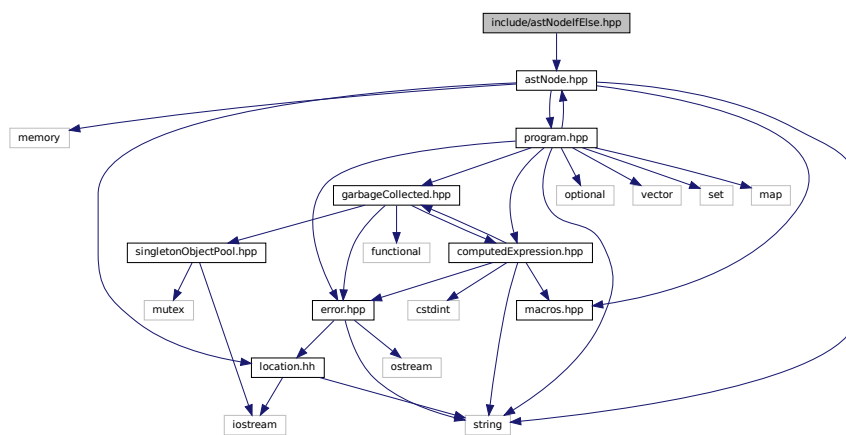
6.16.1 Detailed Description

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

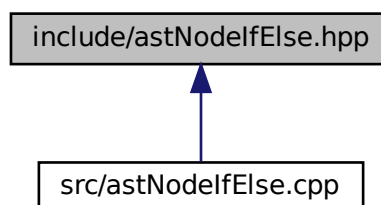
6.17 include/astNodeIfElse.hpp File Reference

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.

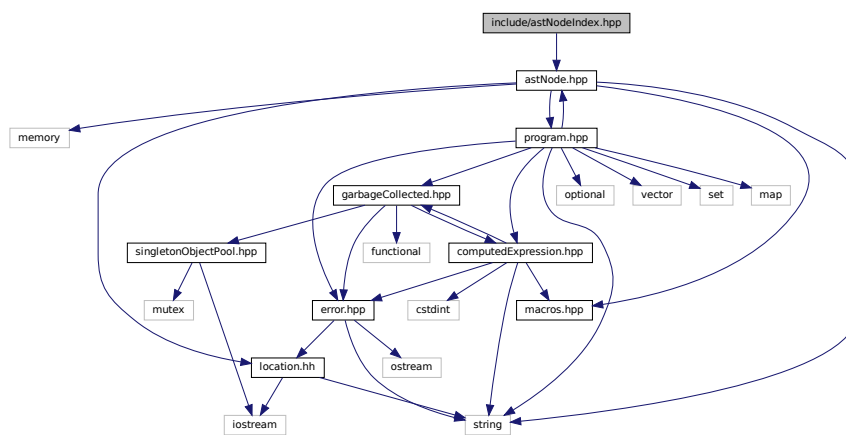
6.17.1 Detailed Description

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

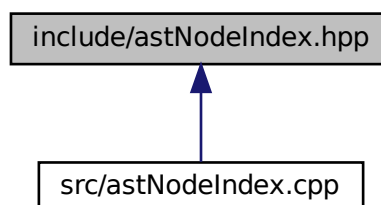
6.18 include/astNodeIndex.hpp File Reference

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.

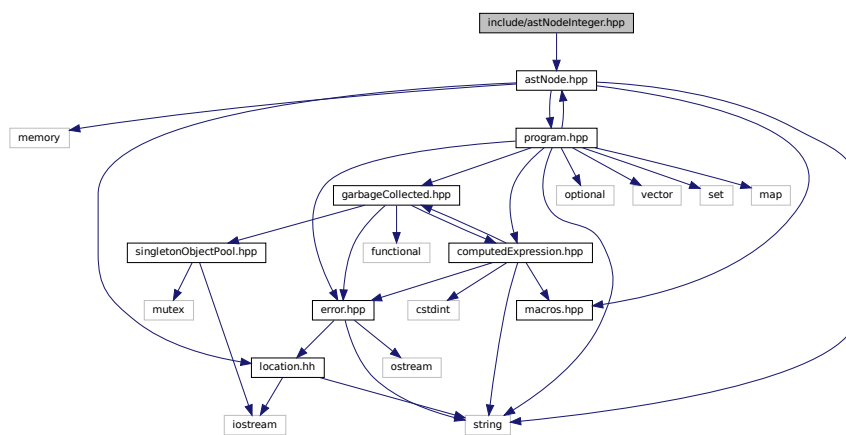
6.18.1 Detailed Description

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

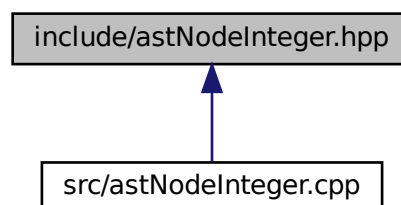
6.19 include/astNodeInteger.hpp File Reference

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

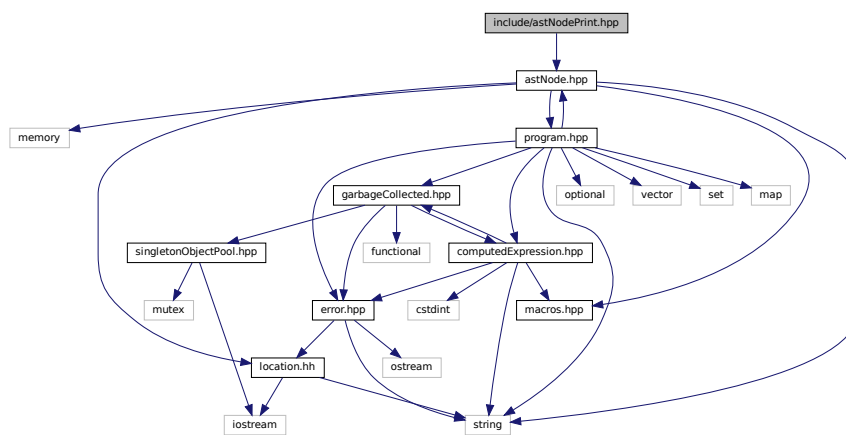
Declare the `Tang::AstNodeInteger` class.

6.20 include/astNodePrint.hpp File Reference

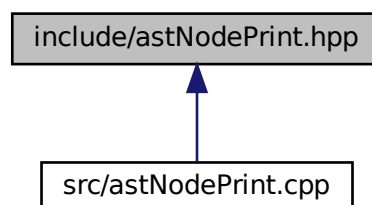
Declare the `Tang::AstNodePrint` class.

```
#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 typeoperation.

6.20.1 Detailed Description

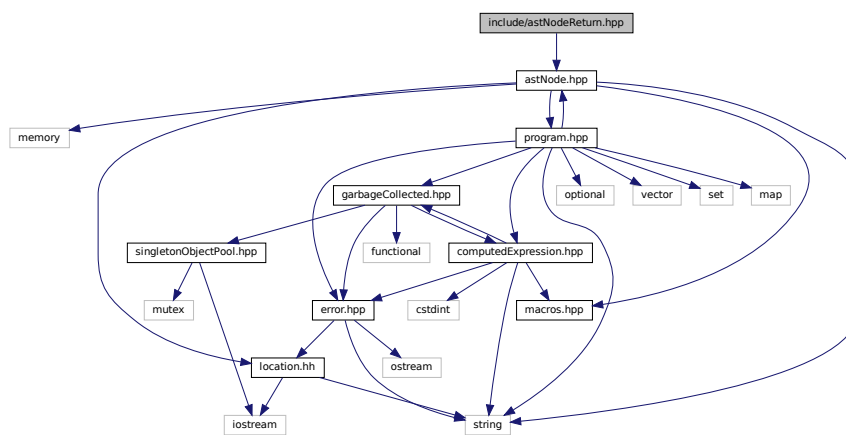
Declare the [Tang::AstNodePrint](#) class.

6.21 include/astNodeReturn.hpp File Reference

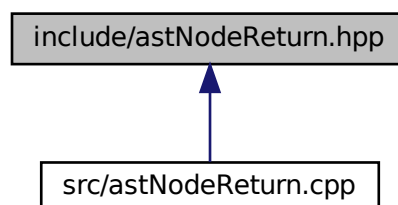
Declare the [Tang::AstNodeReturn](#) class.

```
#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.

6.21.1 Detailed Description

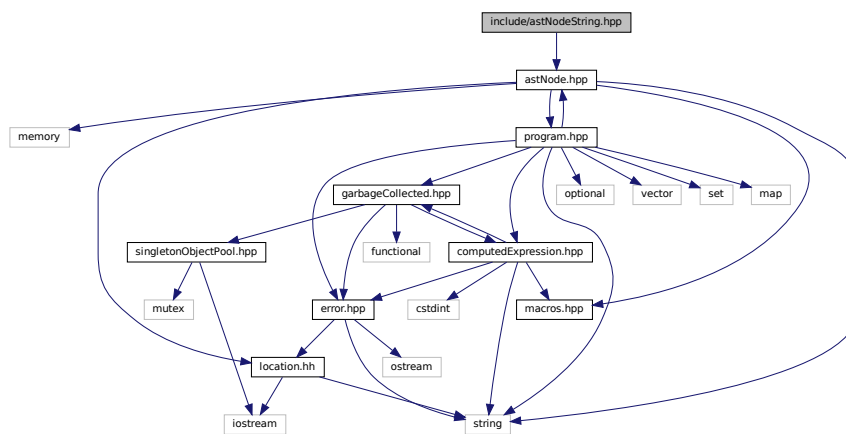
Declare the [Tang::AstNodeReturn](#) class.

6.22 include/astNodeString.hpp File Reference

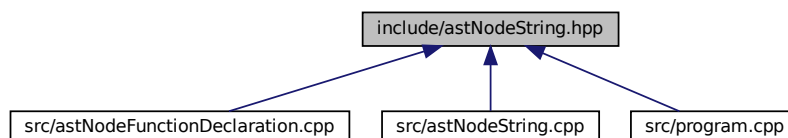
Declare the [Tang::AstNodeString](#) class.

```
#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*.

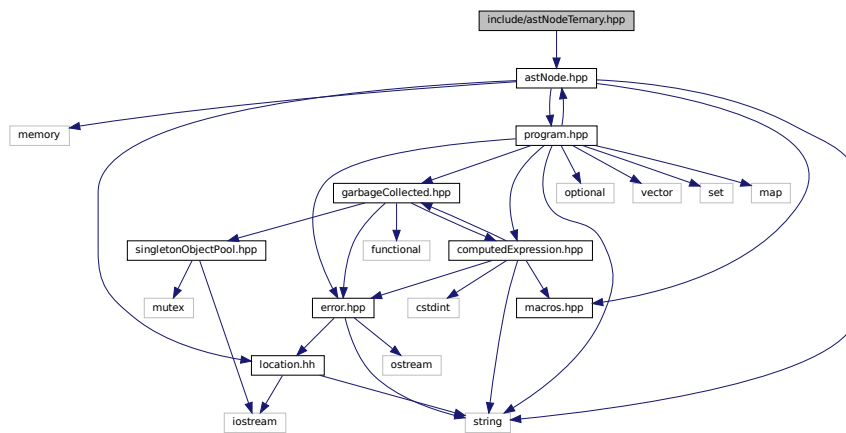
6.22.1 Detailed Description

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

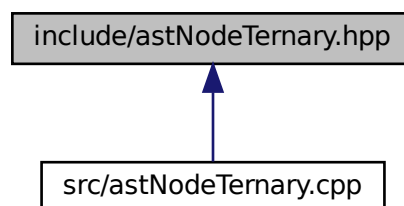
6.23 include/astNodeTernary.hpp File Reference

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.

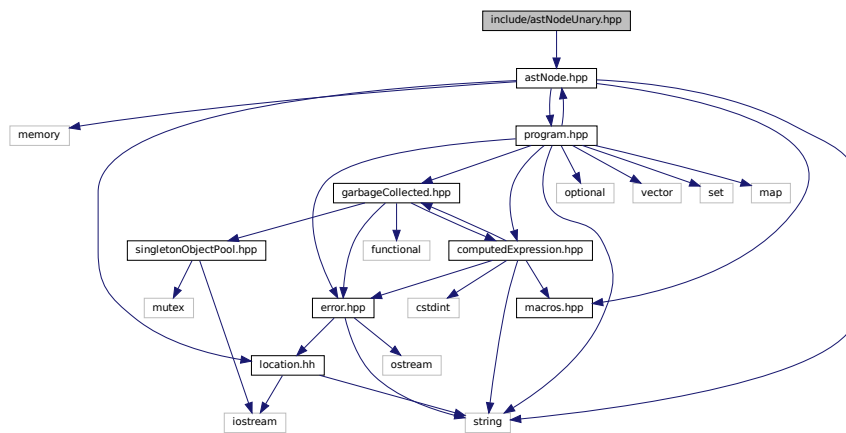
6.23.1 Detailed Description

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

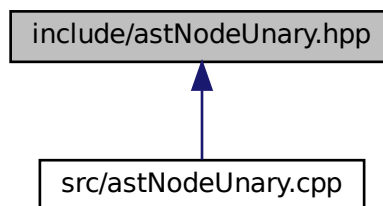
6.24 include/astNodeUnary.hpp File Reference

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.

6.24.1 Detailed Description

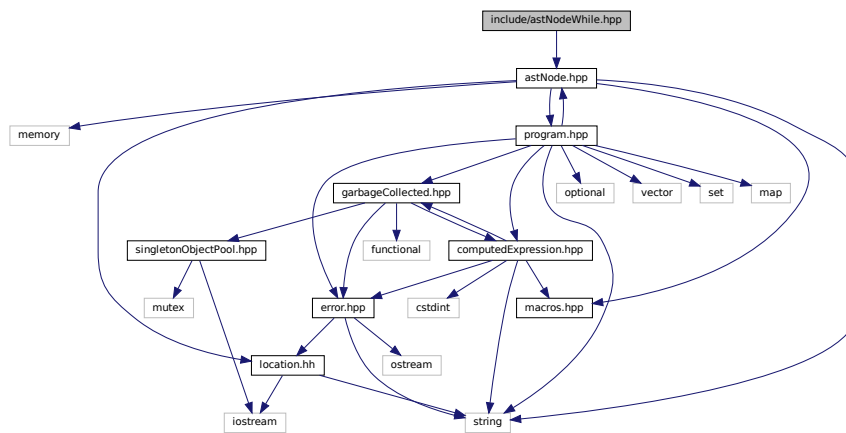
Declare the [Tang::AstNodeUnary](#) class.

6.25 include/astNodeWhile.hpp File Reference

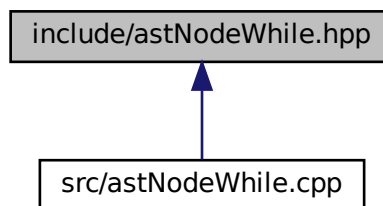
Declare the [Tang::AstNodeWhile](#) class.

```
#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.

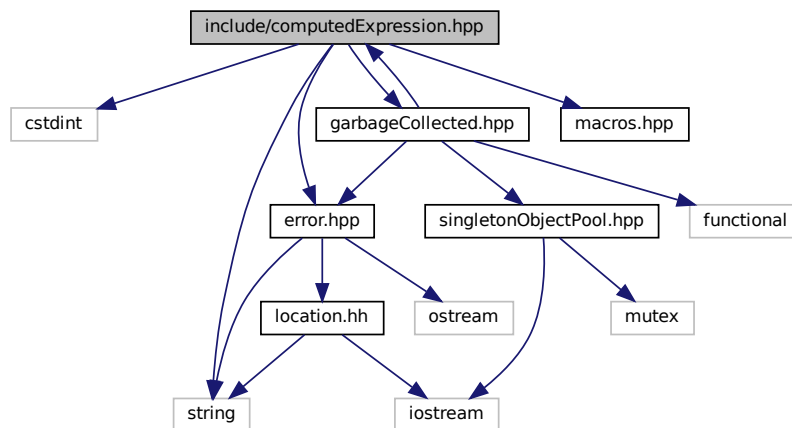
6.25.1 Detailed Description

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

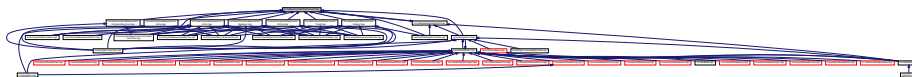
6.26 include/computedExpression.hpp File Reference

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

```
#include <cstdint>
#include <string>
#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.

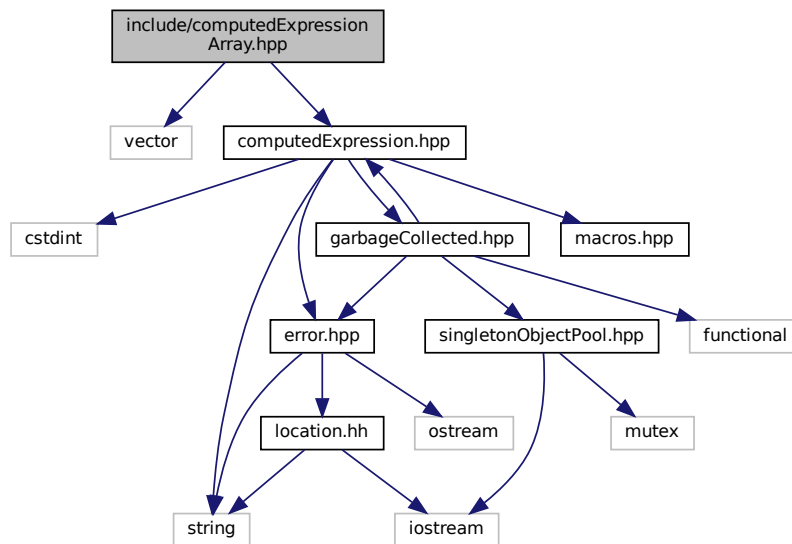
6.26.1 Detailed Description

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

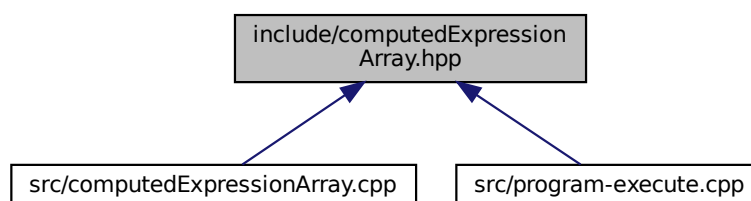
6.27 include/computedExpressionArray.hpp File Reference

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.

6.27.1 Detailed Description

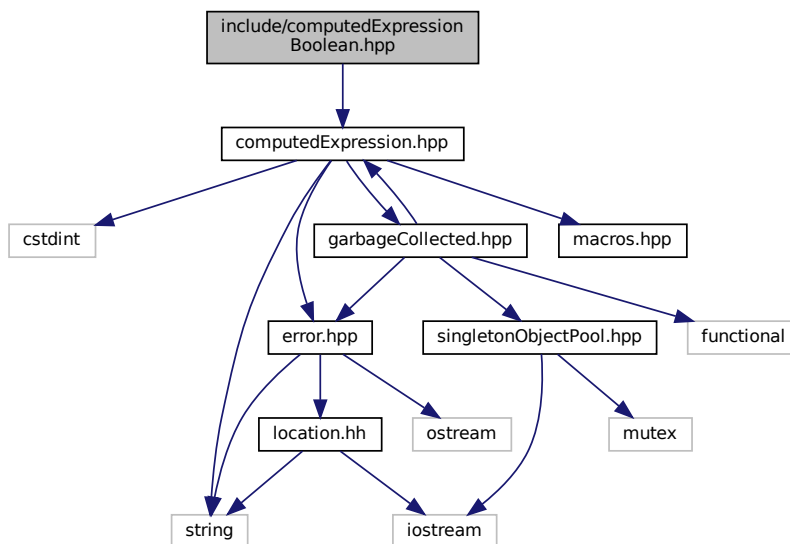
Declare the [Tang::ComputedExpressionArray](#) class.

6.28 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.

6.28.1 Detailed Description

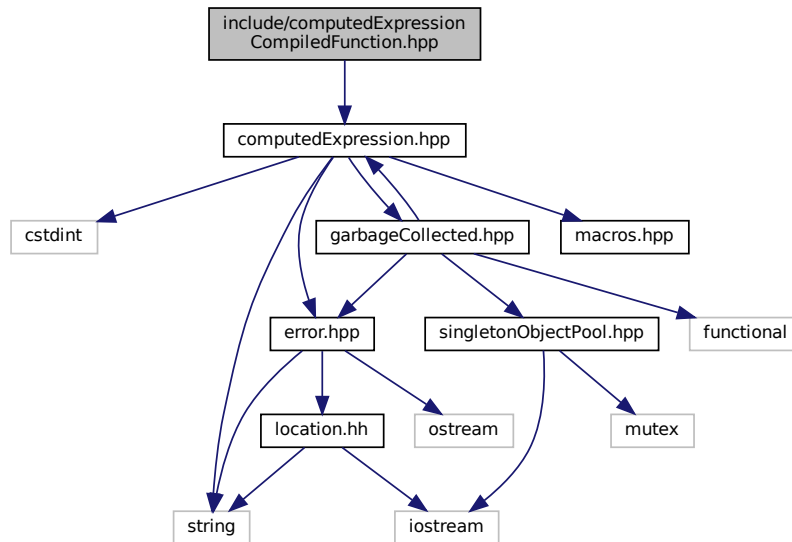
Declare the `Tang::ComputedExpressionBoolean` class.

6.29 include/computedExpressionCompiledFunction.hpp File Reference

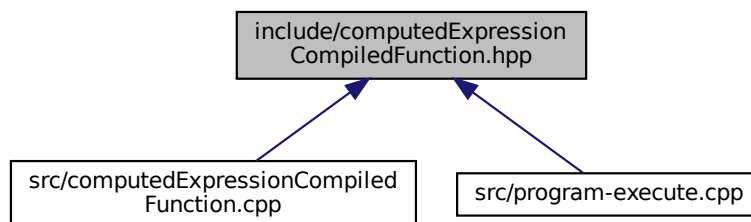
Declare the [Tang::ComputedExpressionCompiledFunction](#) class.

```
#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.

6.29.1 Detailed Description

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

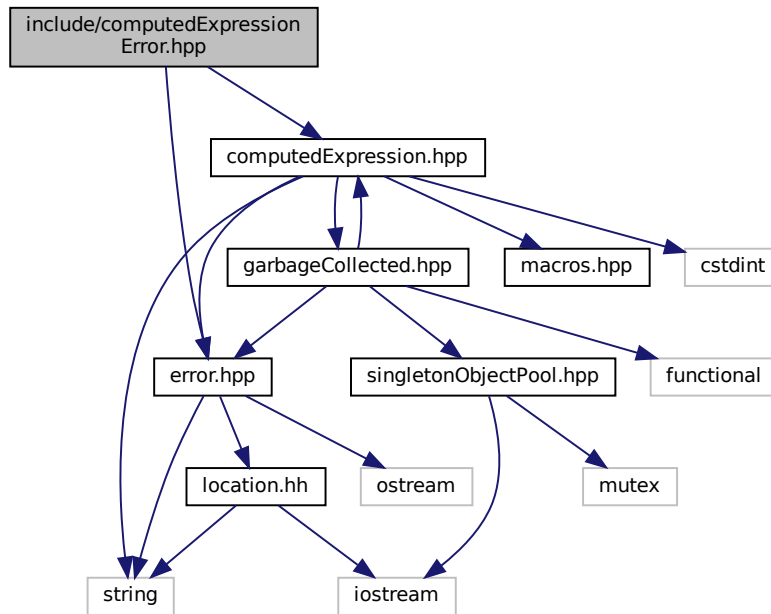
6.30 include/computedExpressionError.hpp File Reference

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

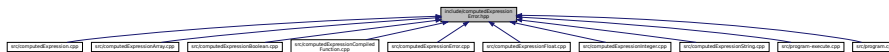
```
#include "computedExpression.hpp"
```

```
#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](#).

6.30.1 Detailed Description

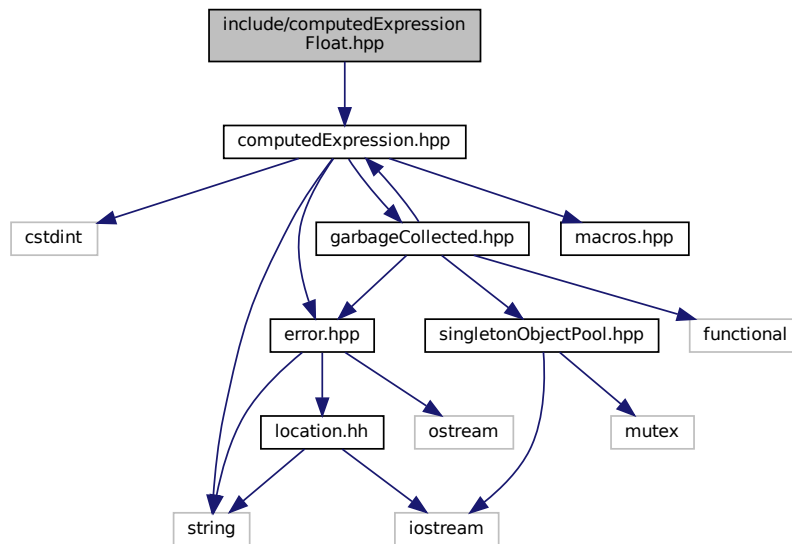
Declare the [Tang::ComputedExpressionError](#) class.

6.31 include/computedExpressionFloat.hpp File Reference

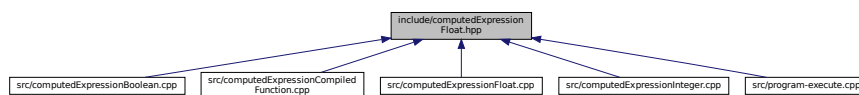
Declare the [Tang::ComputedExpressionFloat](#) class.

```
#include "computedExpression.hpp"
```

Include dependency graph for computedExpressionFloat.hpp:



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



Classes

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

6.31.1 Detailed Description

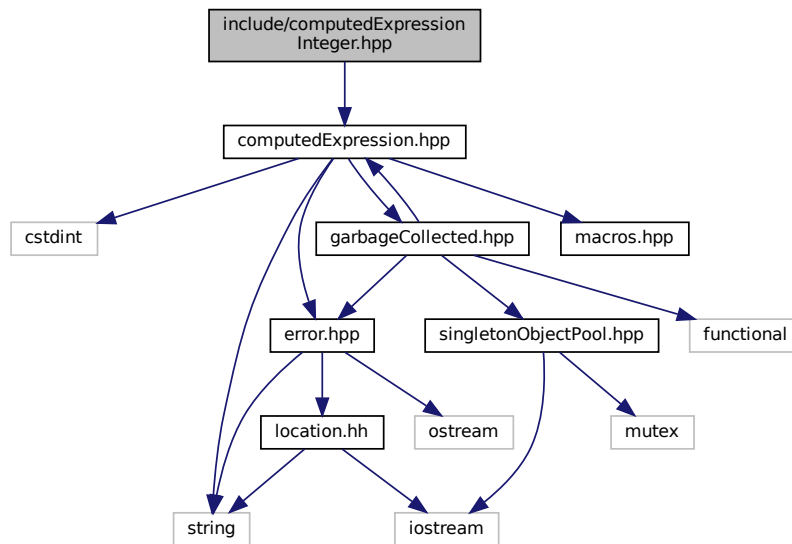
Declare the [Tang::ComputedExpressionFloat](#) class.

6.32 include/computedExpressionInteger.hpp File Reference

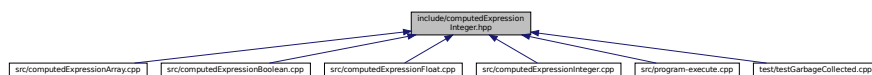
Declare the [Tang::ComputedExpressionInteger](#) class.

```
#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.

6.32.1 Detailed Description

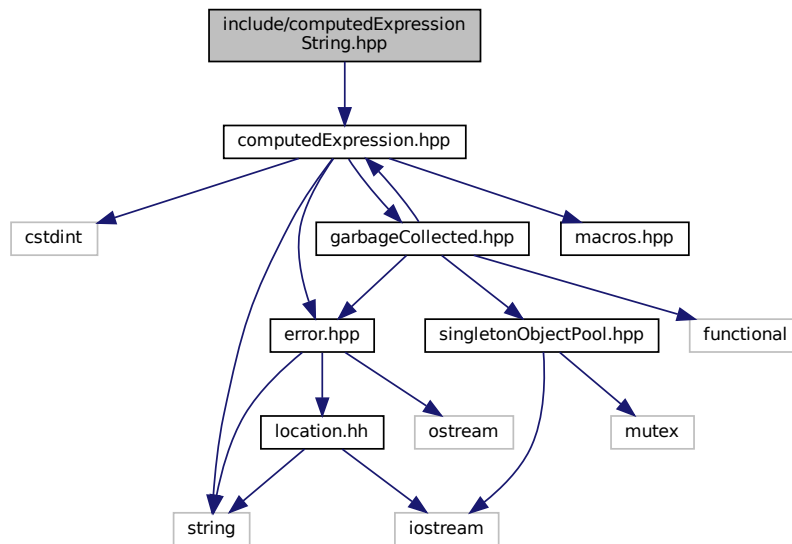
Declare the [Tang::ComputedExpressionInteger](#) class.

6.33 include/computedExpressionString.hpp File Reference

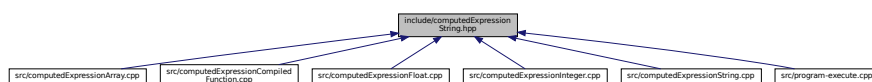
Declare the [Tang::ComputedExpressionString](#) class.

```
#include "computedExpression.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.

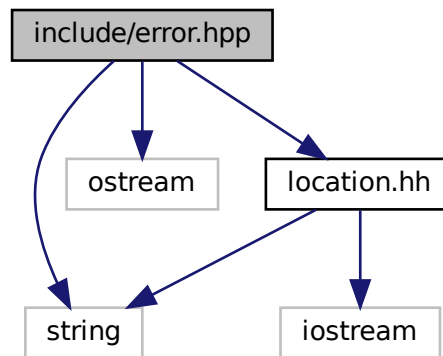
6.33.1 Detailed Description

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

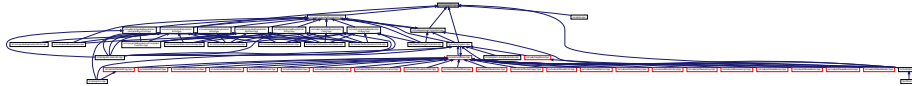
6.34 include/error.hpp File Reference

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

```
#include <string>
#include <ostream>
#include "location.hh"
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.

6.34.1 Detailed Description

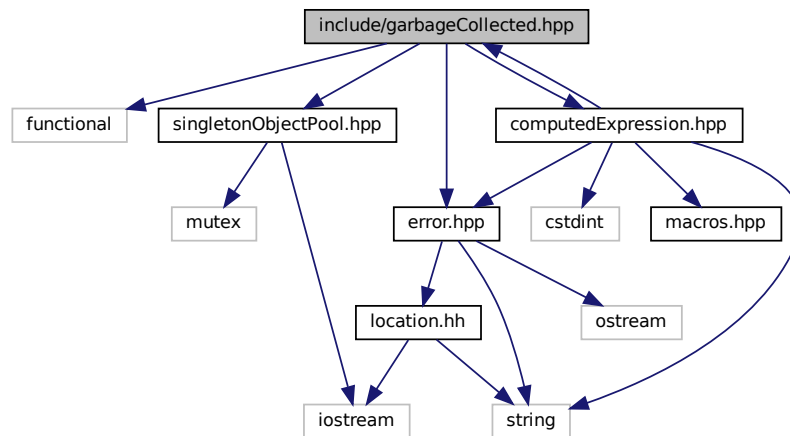
Declare the [Tang::Error](#) class used to describe syntax and runtime errors.

6.35 include/garbageCollected.hpp File Reference

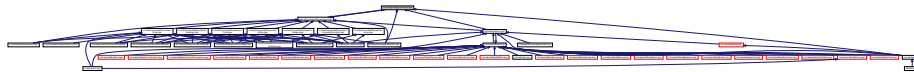
Declare the [Tang::GarbageCollected](#) class.

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

Include dependency graph for garbageCollected.hpp:



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



Classes

- class [Tang::GarbageCollected](#)

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

6.35.1 Detailed Description

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

6.36 include/macros.hpp File Reference

Contains generic macros.

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.

6.36.1 Detailed Description

Contains generic macros.

6.37 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` , `JMP` ,
`JMPF` , `JMPF_POP` , `JMPT` , `JMPT_POP` ,
`NULLVAL` , `INTEGER` , `FLOAT` , `BOOLEAN` ,
`STRING` , `ARRAY` , `FUNCTION` , `ADD` ,
`SUBTRACT` , `MULTIPLY` , `DIVIDE` , `MODULO` ,
`NEGATIVE` , `NOT` , `LT` , `LTE` ,
`GT` , `GTE` , `EQ` , `NEQ` ,
`INDEX` , `CASTINTEGER` , `CASTFLOAT` , `CASTBOOLEAN` ,
`CALLFUNC` , `RETURN` , `PRINT` }

6.37.1 Detailed Description

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

6.37.2 Enumeration Type Documentation

6.37.2.1 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 #.
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.
FUNCTION	Get argc, PC#: push function(argc, PC #)
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.
INDEX	Pop index, pop collection, push collection[index].
CASTINTEGER	Pop a val, typecast to int, push.
CASTFLOAT	Pop a val, typecast to float, push.
CASTBOOLEAN	Pop a val, typecast to boolean, 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.

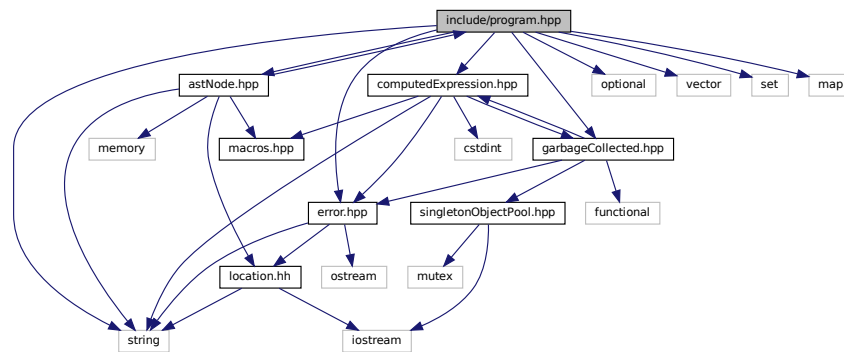
6.38 include/program.hpp File Reference

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 "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.

6.38.1 Detailed Description

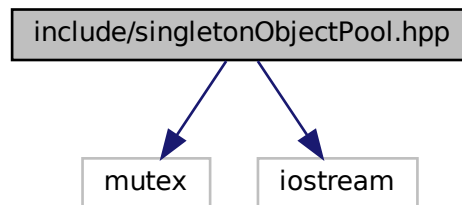
Declare the [Tang::Program](#) class used to compile and execute source code.

6.39 include/singletonObjectPool.hpp File Reference

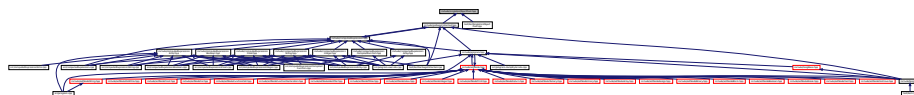
Declare the [Tang::SingletonObjectPool](#) class.

```
#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.

6.39.1 Detailed Description

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

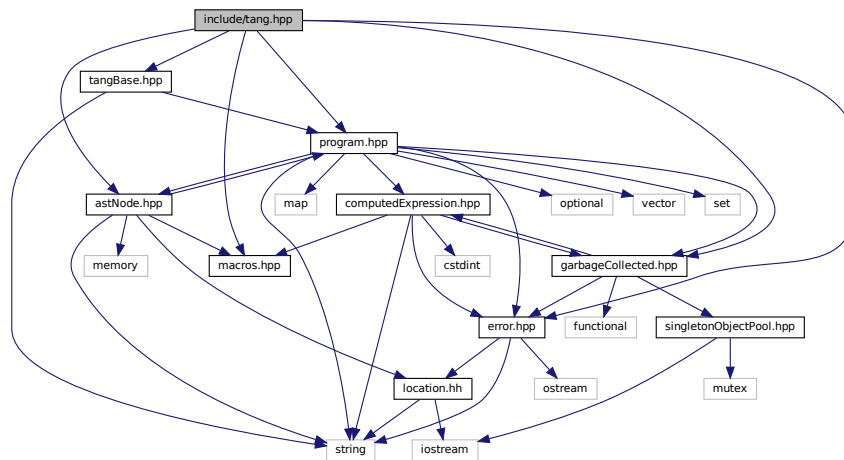
6.40 include/tang.hpp File Reference

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

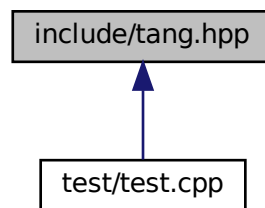
```
#include "macros.hpp"
#include "tangBase.hpp"
#include "astNode.hpp"
#include "error.hpp"
#include "garbageCollected.hpp"
```

```
#include "program.hpp"
```

Include dependency graph for tang.hpp:



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



6.40.1 Detailed Description

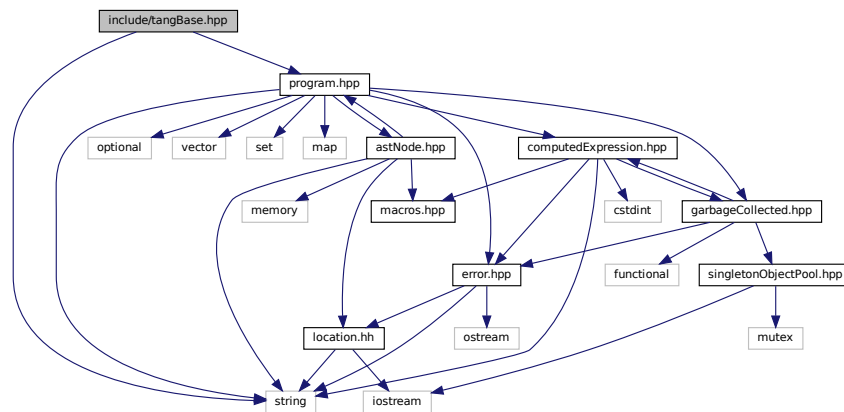
Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

6.41 include/tangBase.hpp File Reference

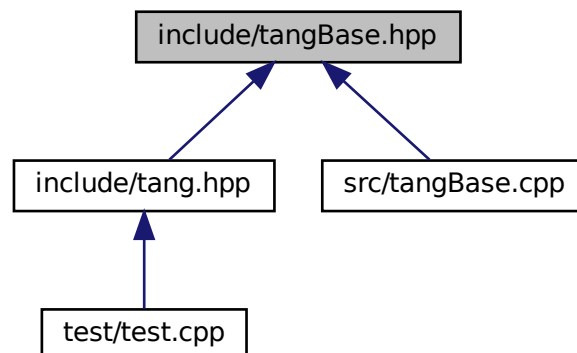
Declare the [Tang::TangBase](#) class used to interact with Tang.

```
#include <string>
#include "program.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.

6.41.1 Detailed Description

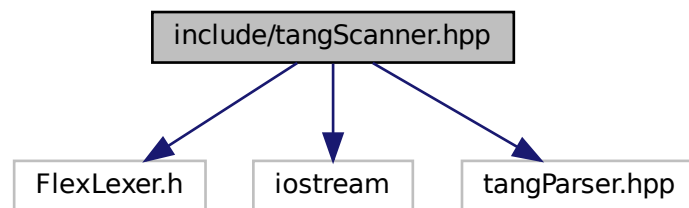
Declare the [Tang::TangBase](#) class used to interact with Tang.

6.42 include/tangScanner.hpp File Reference

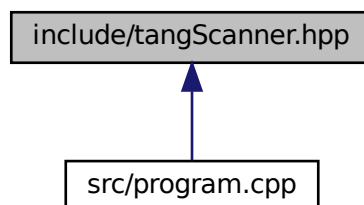
Declare the [Tang::TangScanner](#) used to tokenize a Tang script.

```
#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\(\)](#)

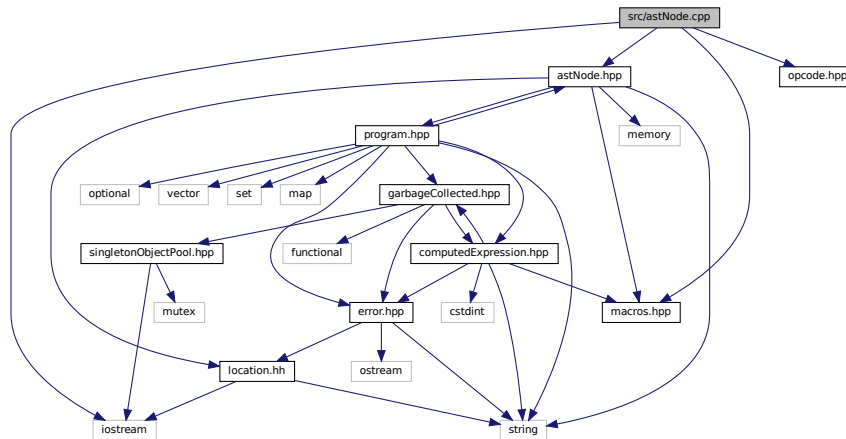
6.42.1 Detailed Description

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

6.43 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:
```



6.43.1 Detailed Description

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

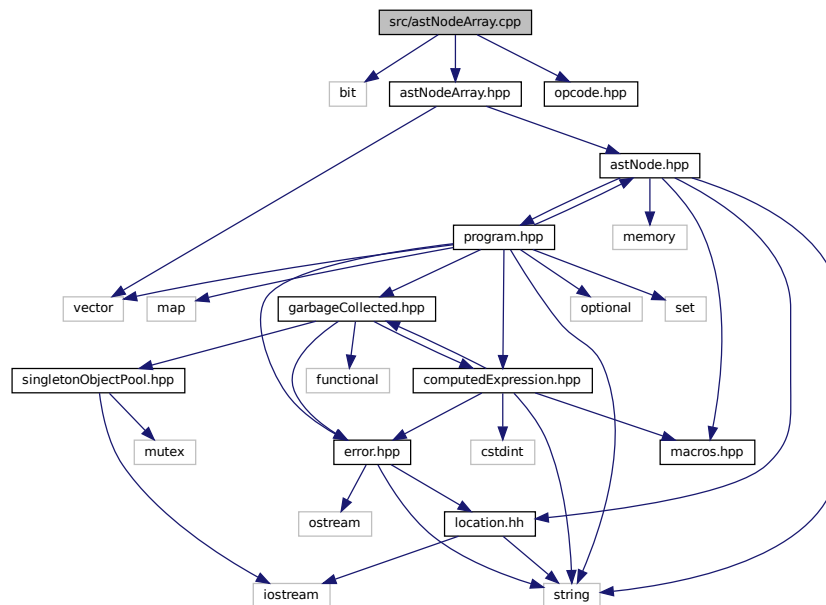
6.44 src/astNodeArray.cpp File Reference

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

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

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

Include dependency graph for astNodeArray.cpp:



6.44.1 Detailed Description

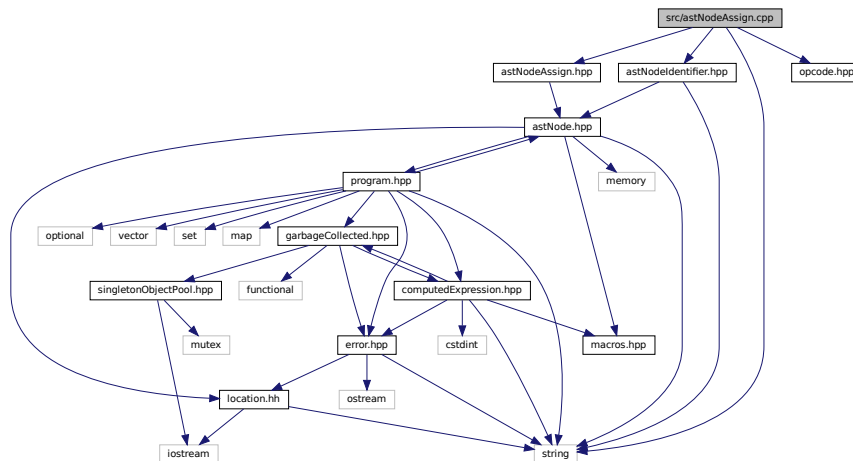
Define the [Tang::AstNodeArray](#) class.

6.45 src/astNodeAssign.cpp File Reference

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

```
#include <string>
#include "astNodeAssign.hpp"
#include "astNodeIdentifier.hpp"
#include "opcode.hpp"
```

Include dependency graph for `astNodeAssign.cpp`:



6.45.1 Detailed Description

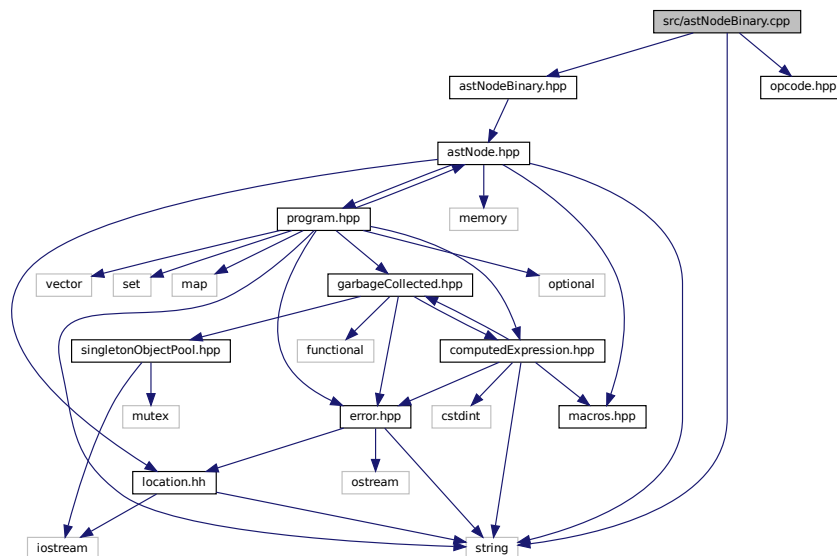
Define the [Tang::AstNodeAssign](#) class.

6.46 src/astNodeBinary.cpp File Reference

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

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

Include dependency graph for `astNodeBinary.cpp`:



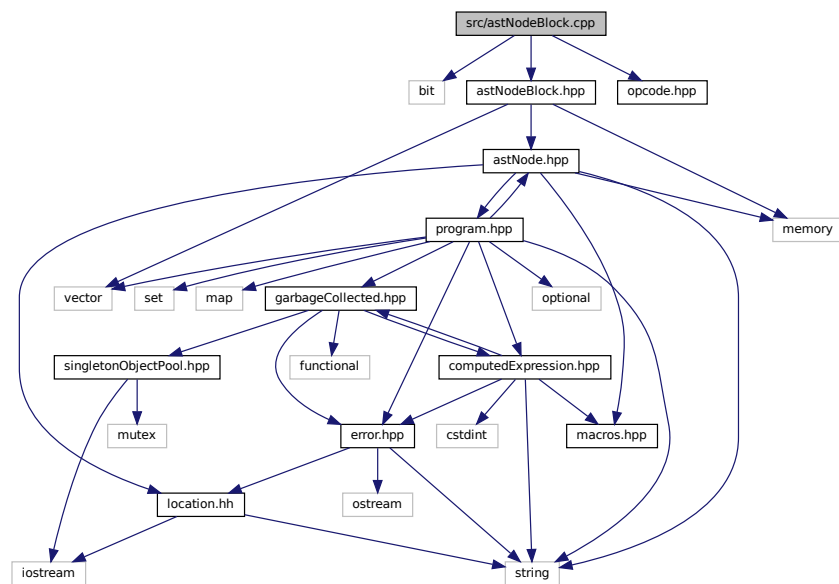
6.46.1 Detailed Description

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

6.47 src/astNodeBlock.cpp File Reference

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

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



6.47.1 Detailed Description

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

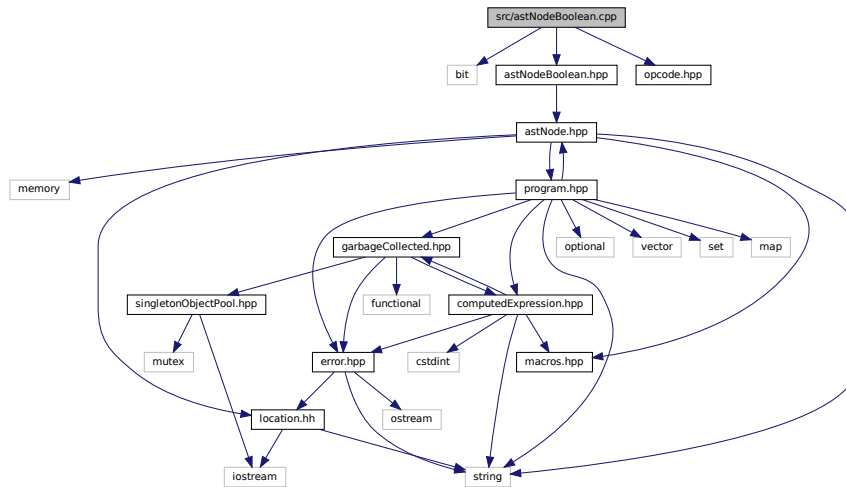
6.48 src/astNodeBoolean.cpp File Reference

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

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

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

Include dependency graph for astNodeBoolean.cpp:



6.48.1 Detailed Description

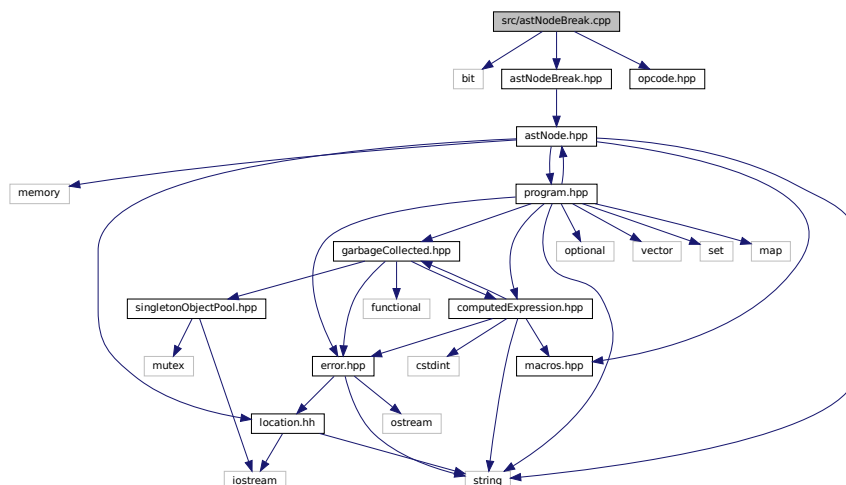
Define the [Tang::AstNodeBoolean](#) class.

6.49 src/astNodeBreak.cpp File Reference

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

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

Include dependency graph for astNodeBreak.cpp:



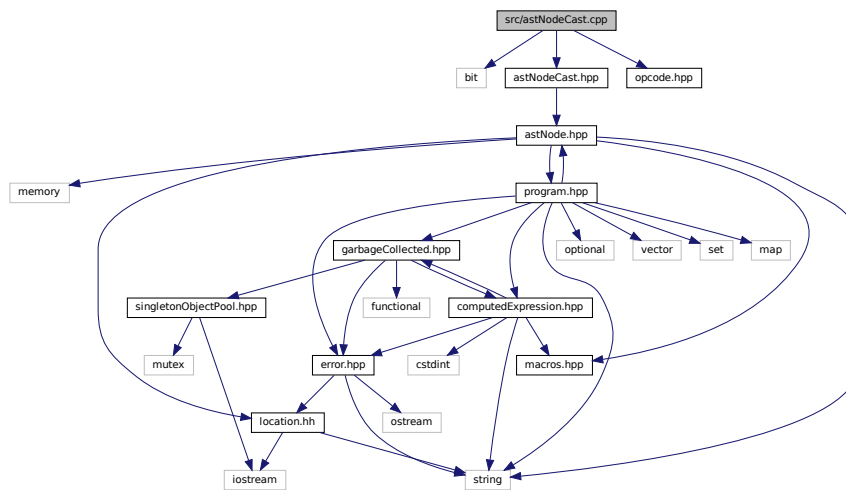
6.49.1 Detailed Description

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

6.50 src/astNodeCast.cpp File Reference

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

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



6.50.1 Detailed Description

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

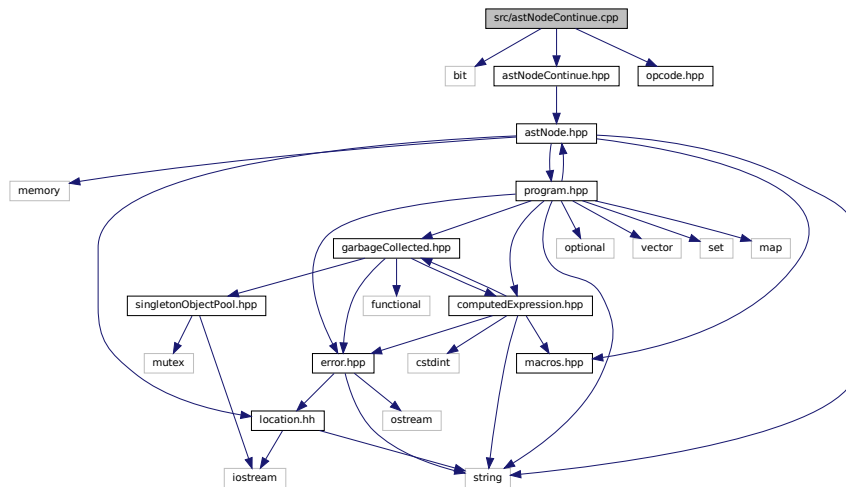
6.51 src/astNodeContinue.cpp File Reference

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

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

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

Include dependency graph for `astNodeContinue.cpp`:



6.51.1 Detailed Description

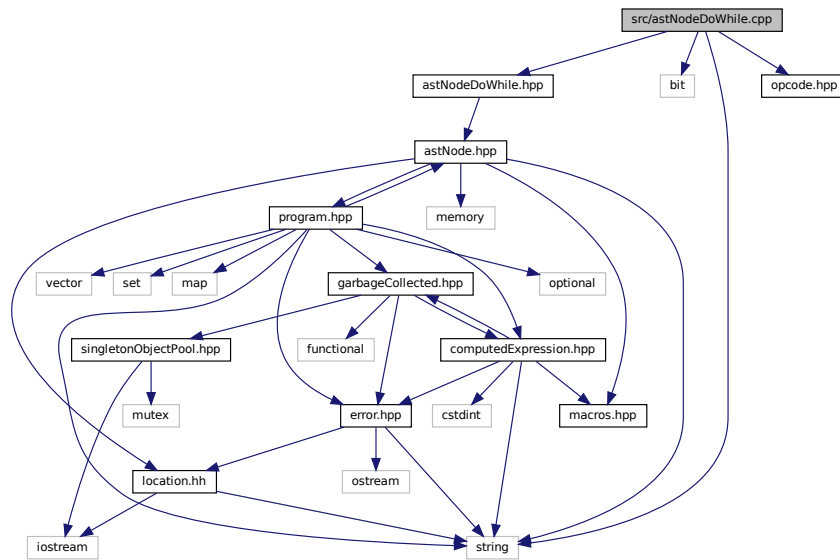
Define the [Tang::AstNodeContinue](#) class.

6.52 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:



6.52.1 Detailed Description

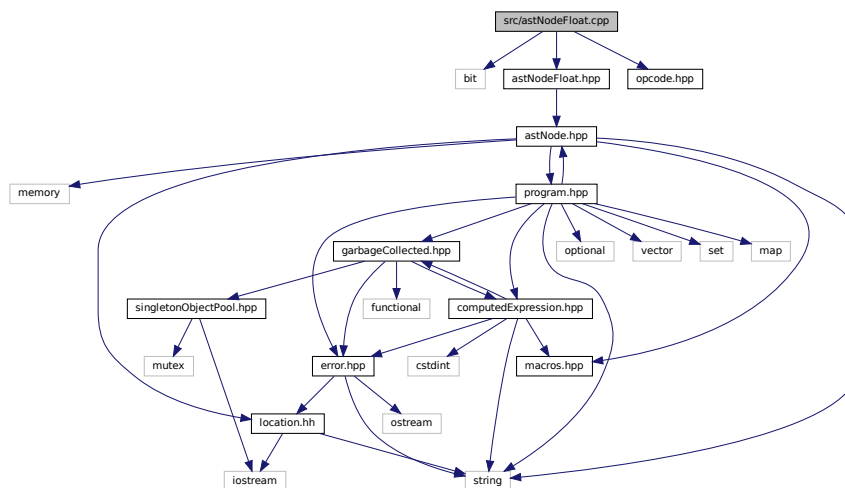
Define the [Tang::AstNodeDoWhile](#) class.

6.53 src/astNodeFloat.cpp File Reference

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

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

Include dependency graph for astNodeFloat.cpp:



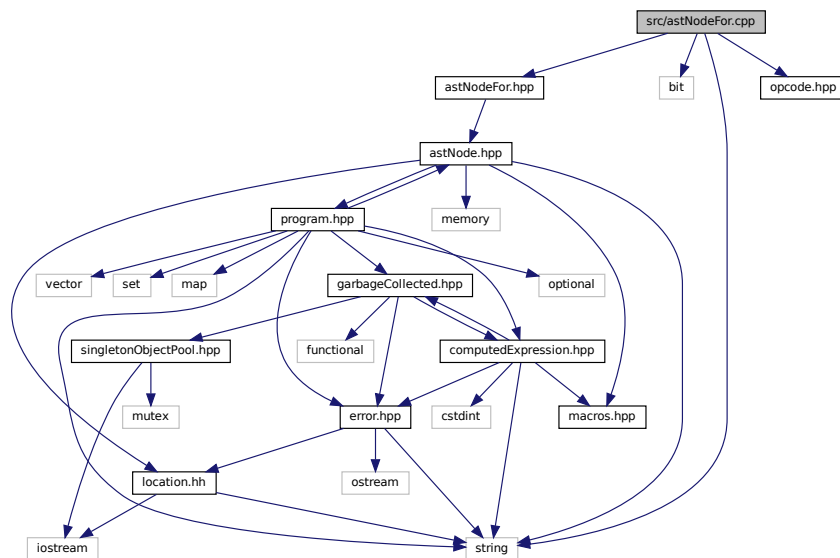
6.53.1 Detailed Description

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

6.54 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:
```



6.54.1 Detailed Description

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

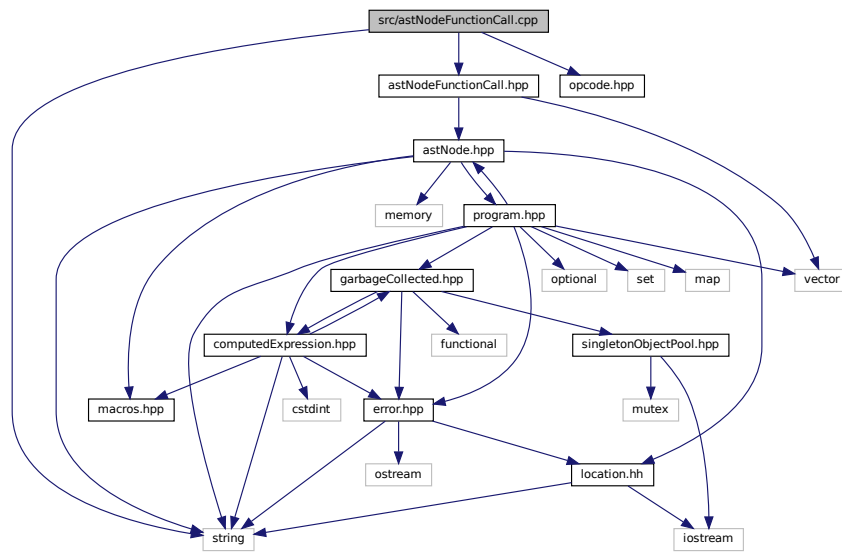
6.55 src/astNodeFunctionCall.cpp File Reference

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

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

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

Include dependency graph for astNodeFunctionCall.cpp:



6.55.1 Detailed Description

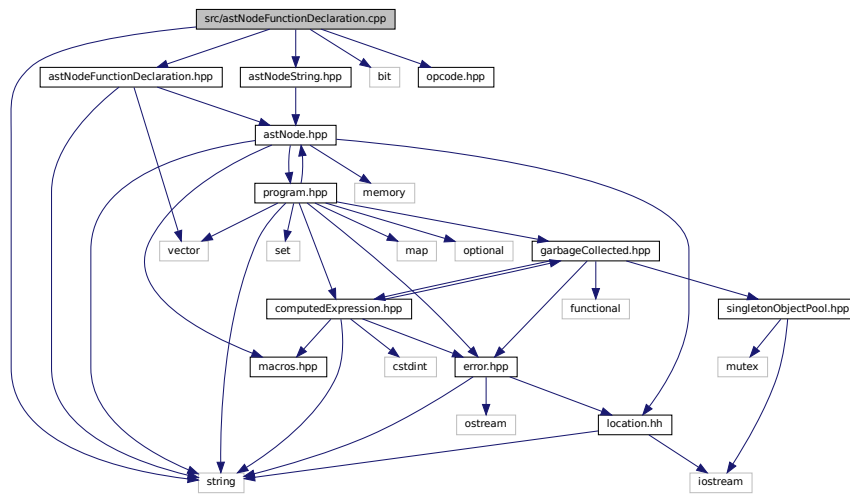
Define the [Tang::AstNodeFunctionCall](#) class.

6.56 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`:



6.56.1 Detailed Description

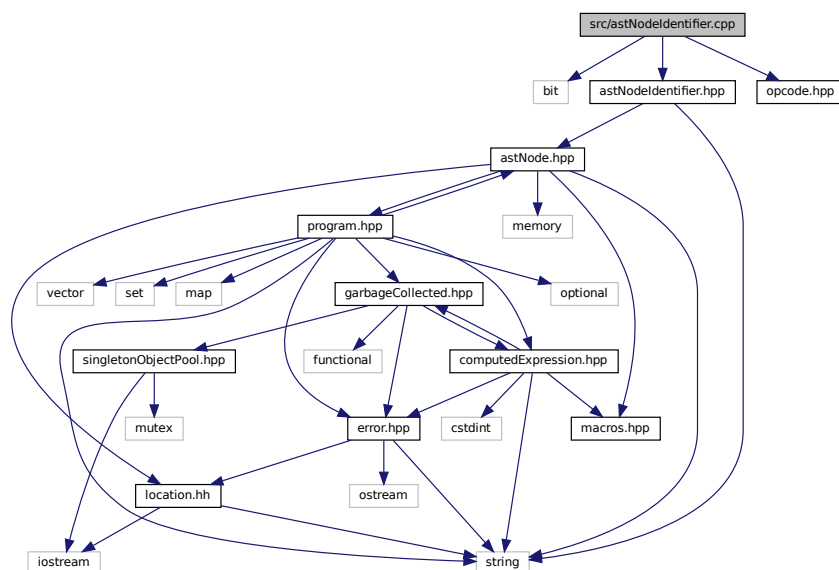
Define the `Tang::AstNodeFunctionDeclaration` class.

6.57 src/astNodeIdentifier.cpp File Reference

Define the `Tang::AstNodeIdentifier` class.

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

Include dependency graph for `astNodeIdentifier.cpp`:



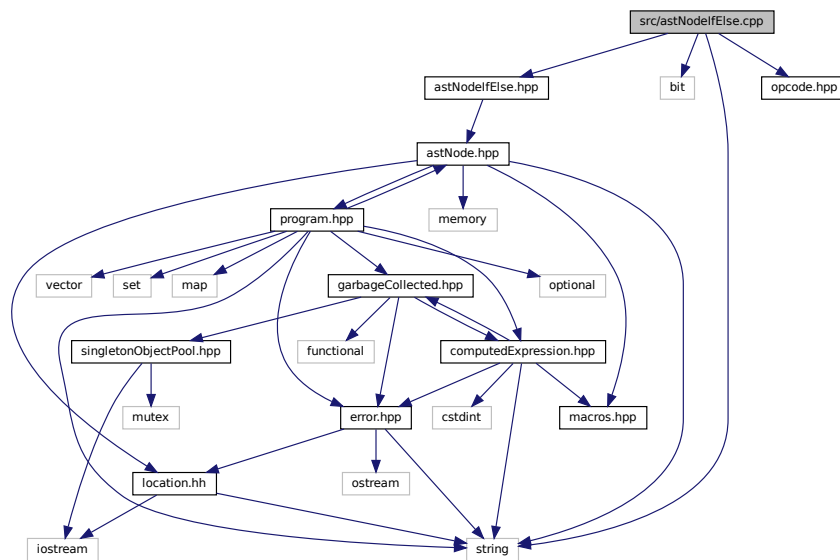
6.57.1 Detailed Description

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

6.58 src/astNodeIfElse.cpp File Reference

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

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



6.58.1 Detailed Description

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

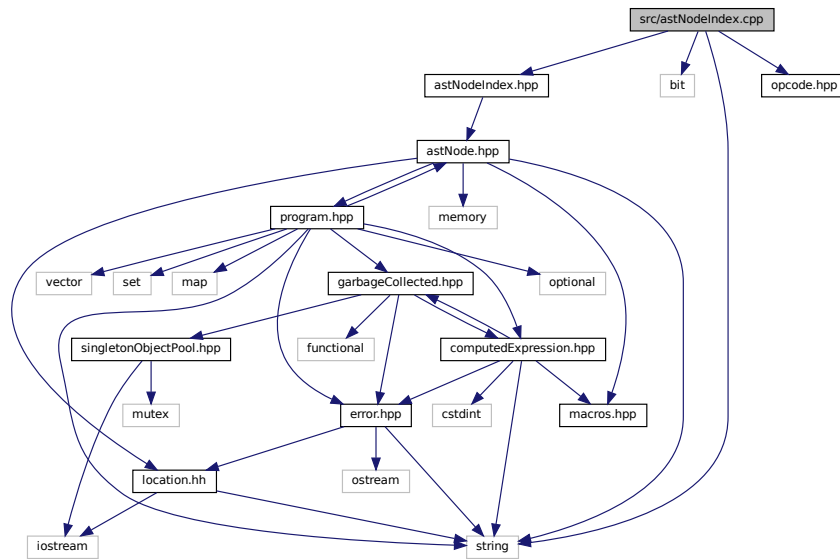
6.59 src/astNodeIndex.cpp File Reference

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

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

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

Include dependency graph for astNodeIndex.cpp:



6.59.1 Detailed Description

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

6.60 src/astNodeInteger.cpp File Reference

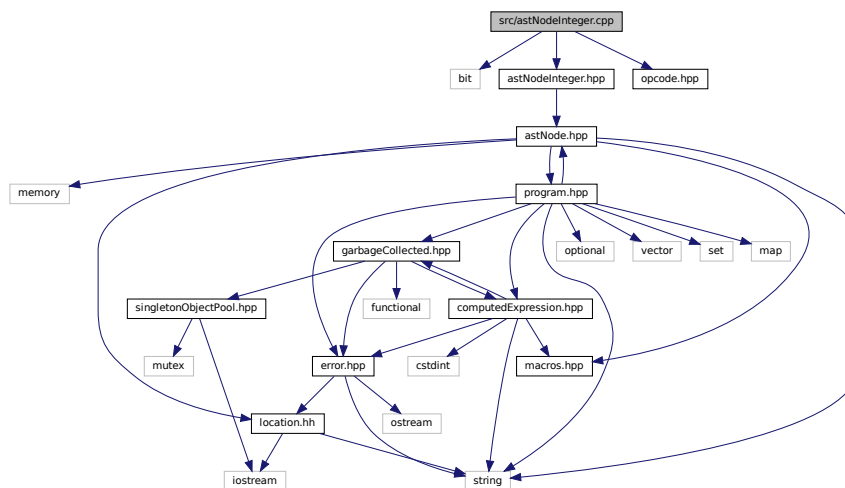
Define the [Tang::AstNodeInteger](#) class.

```
#include <bit>
```

```
#include "astNodeInteger.hpp"
```

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

Include dependency graph for astNodeInteger.cpp:



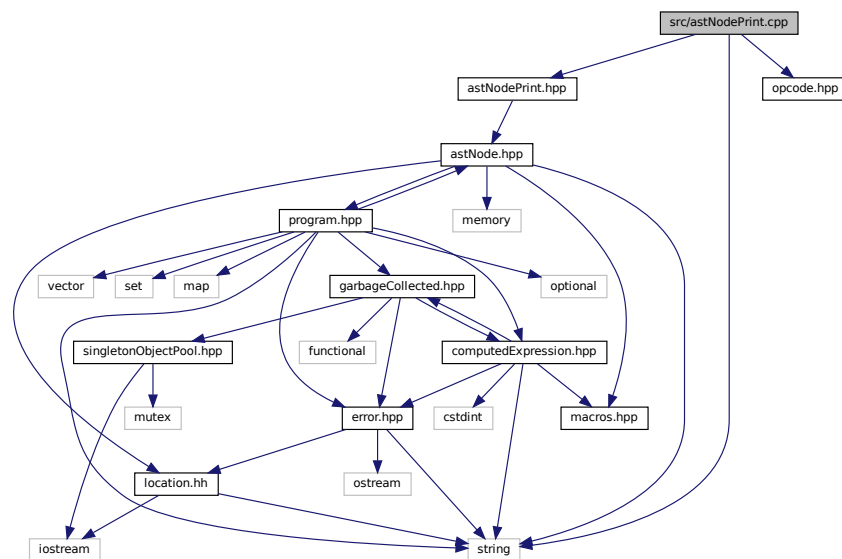
6.60.1 Detailed Description

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

6.61 src/astNodePrint.cpp File Reference

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

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



6.61.1 Detailed Description

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

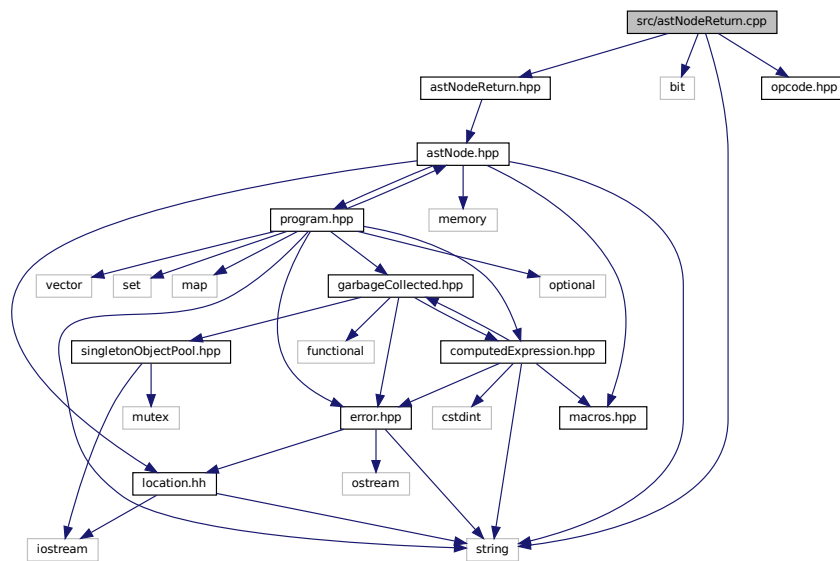
6.62 src/astNodeReturn.cpp File Reference

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

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

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

Include dependency graph for astNodeReturn.cpp:



6.62.1 Detailed Description

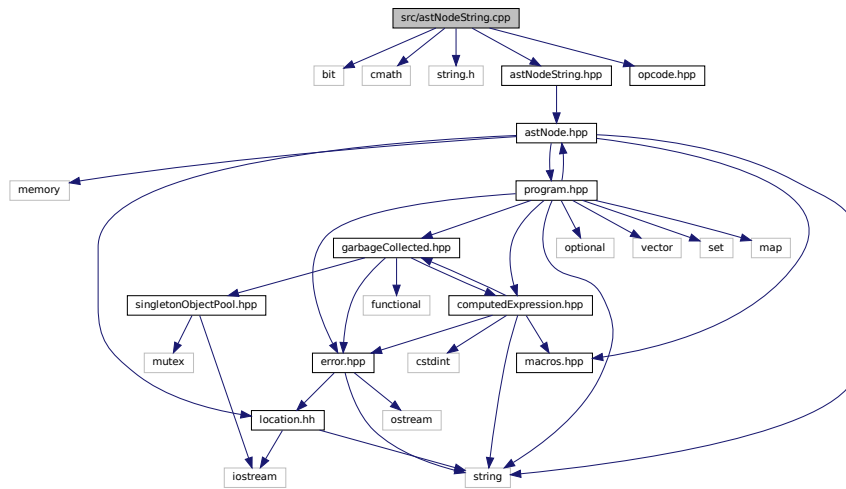
Define the [Tang::AstNodeReturn](#) class.

6.63 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:



6.63.1 Detailed Description

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

6.64 src/astNodeTernary.cpp File Reference

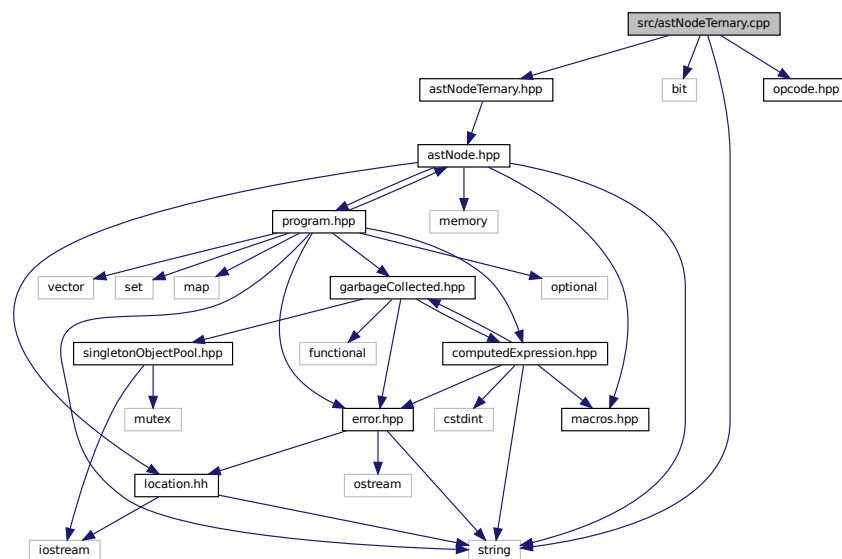
Define the [Tang::AstNodeTernary](#) class.

```

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

```

Include dependency graph for astNodeTernary.cpp:



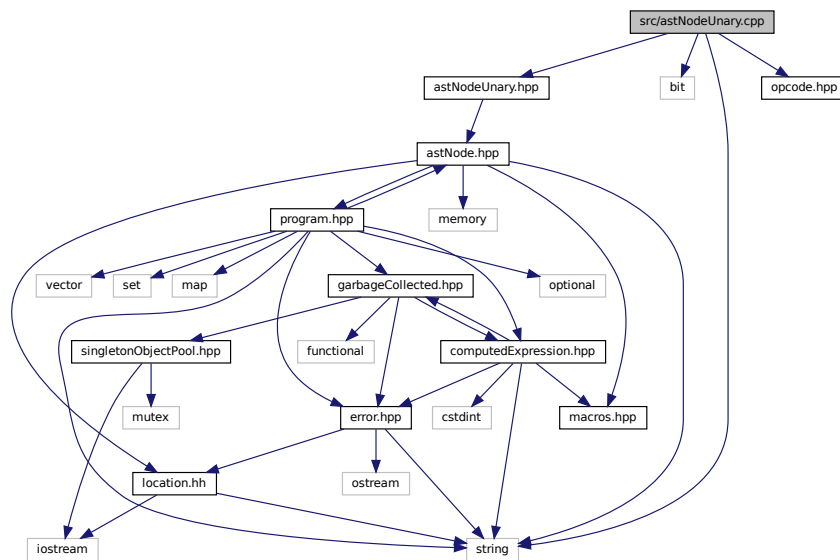
6.64.1 Detailed Description

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

6.65 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:
```



6.65.1 Detailed Description

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

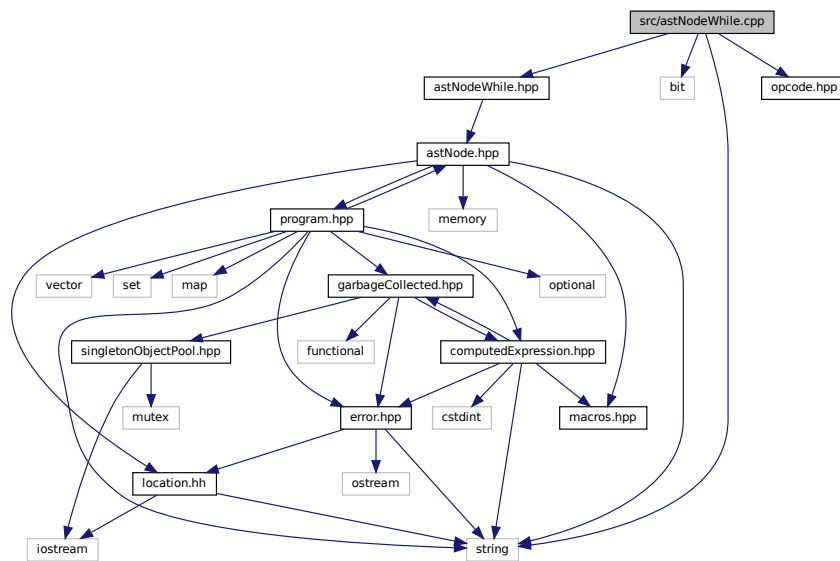
6.66 src/astNodeWhile.cpp File Reference

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

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

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

Include dependency graph for astNodeWhile.cpp:



6.66.1 Detailed Description

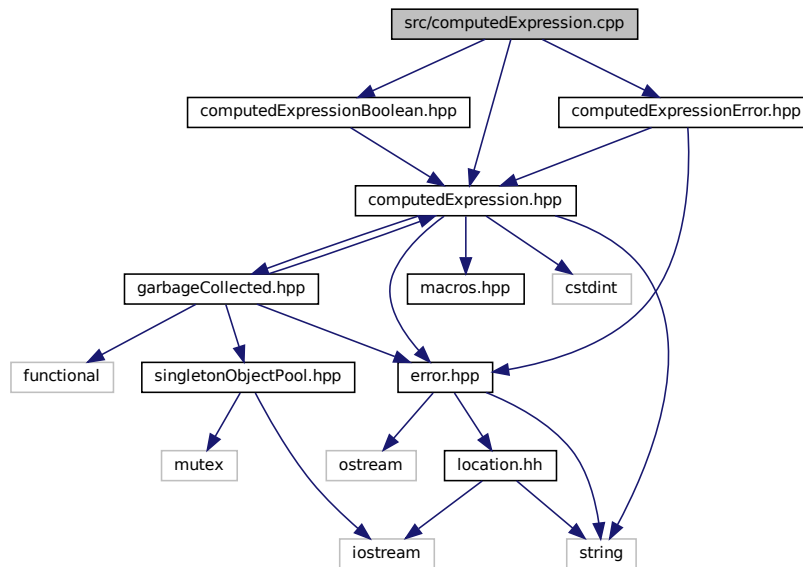
Define the [Tang::AstNodeWhile](#) class.

6.67 src/computedExpression.cpp File Reference

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

```
#include "computedExpression.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionError.hpp"
```

Include dependency graph for `computedExpression.cpp`:



6.67.1 Detailed Description

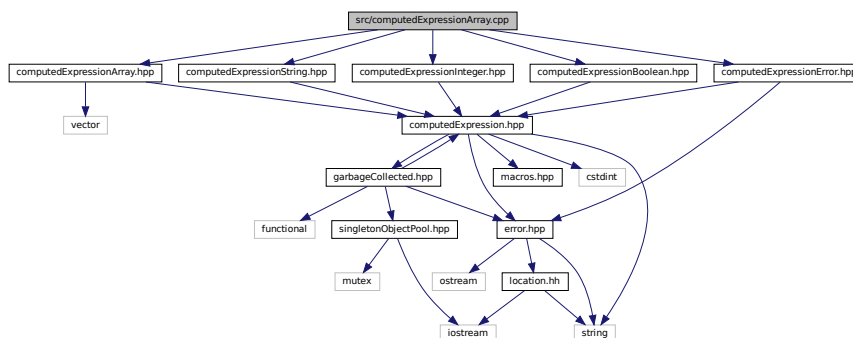
Define the [Tang::ComputedExpression](#) class.

6.68 src/computedExpressionArray.cpp File Reference

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

```
#include "computedExpressionArray.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionError.hpp"
```

Include dependency graph for `computedExpressionArray.cpp`:



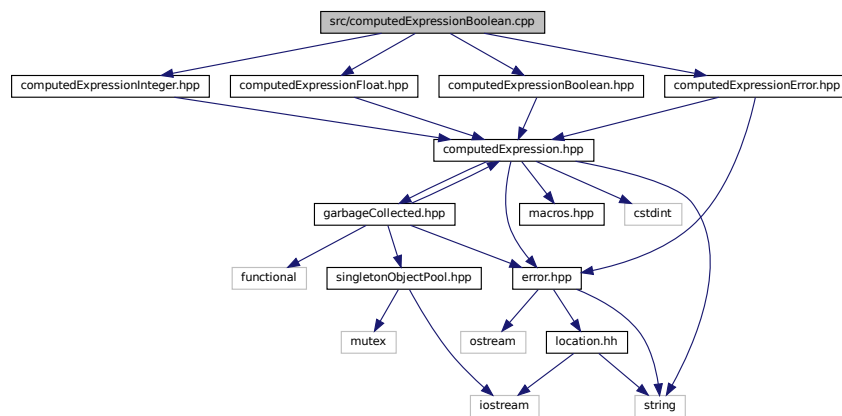
6.68.1 Detailed Description

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

6.69 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:
```



6.69.1 Detailed Description

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

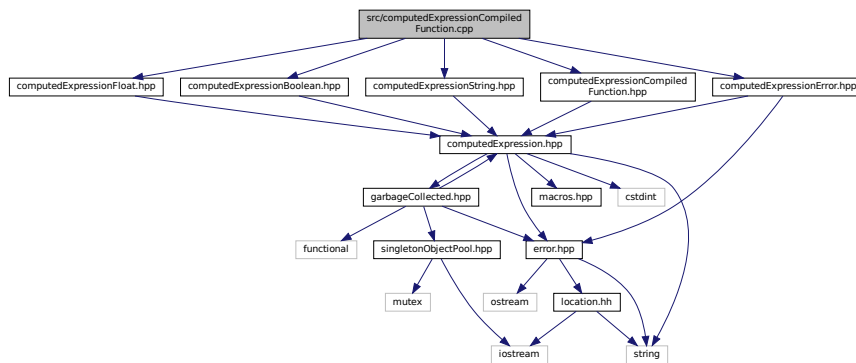
6.70 src/computedExpressionCompiledFunction.cpp File Reference

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`:



6.70.1 Detailed Description

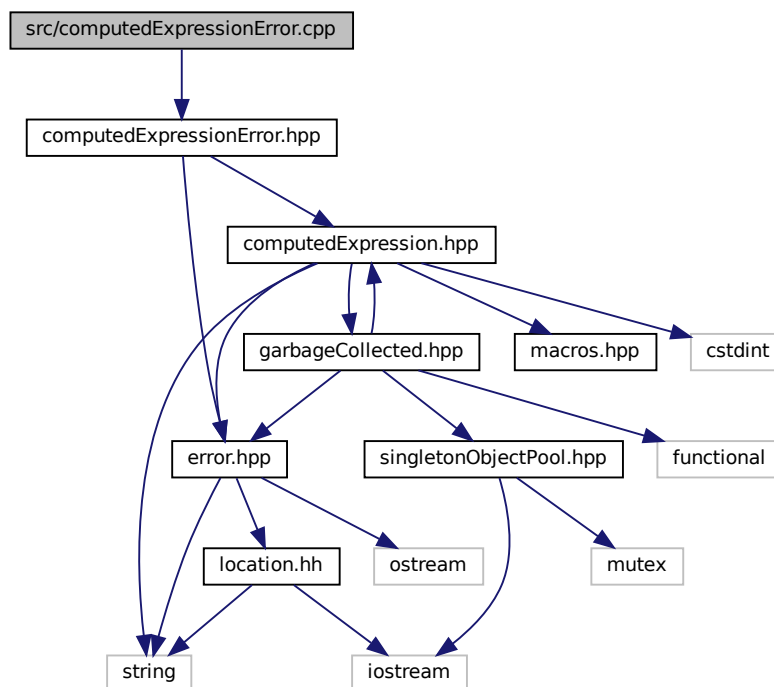
Define the [Tang::ComputedExpressionCompiledFunction](#) class.

6.71 src/computedExpressionError.cpp File Reference

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

```
#include "computedExpressionError.hpp"
```

Include dependency graph for `computedExpressionError.cpp`:



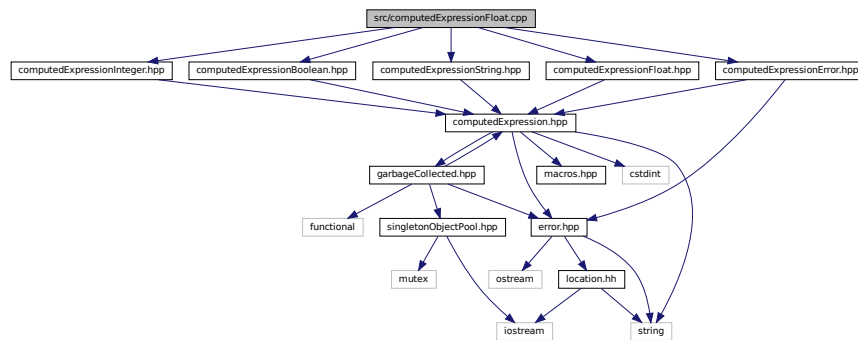
6.71.1 Detailed Description

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

6.72 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:
```



6.72.1 Detailed Description

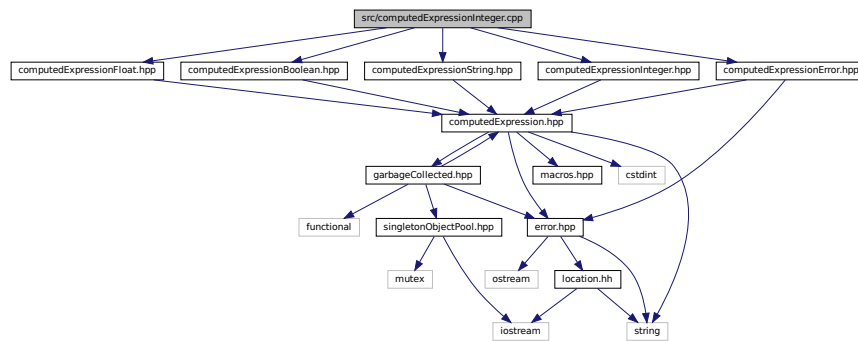
Define the [Tang::ComputedExpressionFloat](#) class.

6.73 src/computedExpressionInteger.cpp File Reference

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

```
#include "computedExpressionInteger.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
```

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



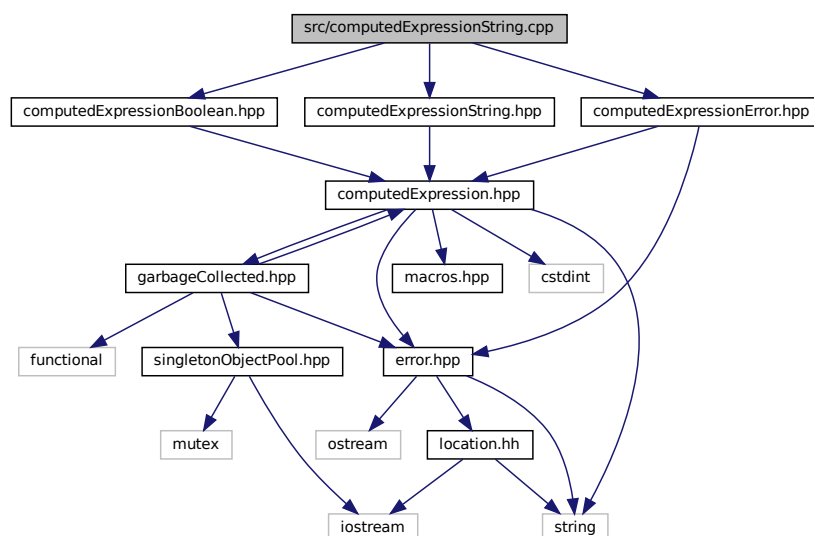
6.73.1 Detailed Description

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

6.74 src/computedExpressionString.cpp File Reference

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

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



6.74.1 Detailed Description

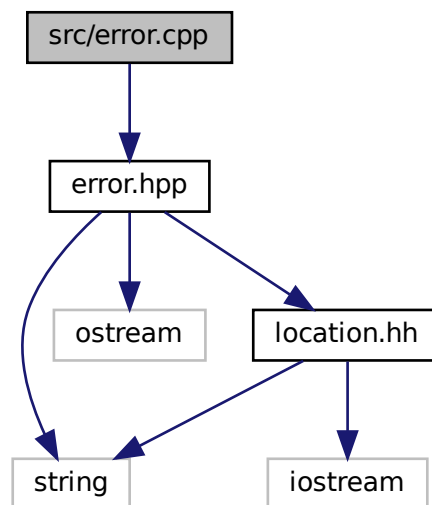
Define the [Tang::ComputedExpressionString](#) class.

6.75 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)`

6.75.1 Detailed Description

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

6.75.2 Function Documentation

6.75.2.1 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

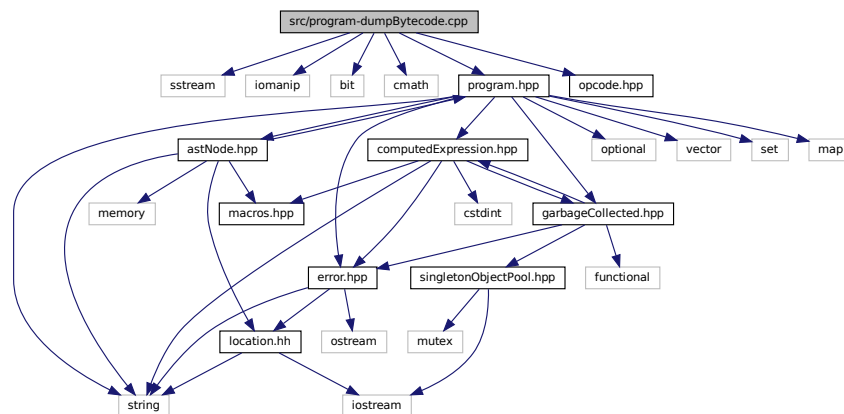
The output stream.

6.76 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"
```

Include dependency graph for program-dumpBytecode.cpp:



Macros

- `#define DUMPPROGRAMCHECK(x)`
Verify the size of the Bytecode vector so that it may be safely accessed.

6.76.1 Detailed Description

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

6.76.2 Macro Definition Documentation

6.76.2.1 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

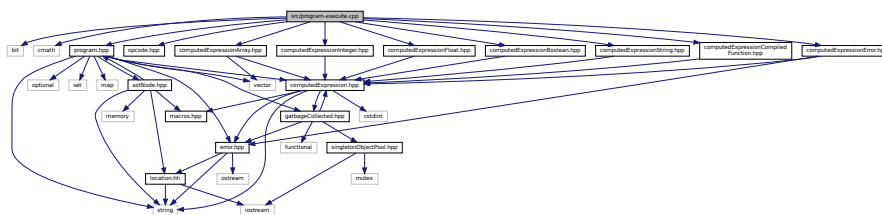
x	The number of additional vector entries that should exist.
---	--

6.77 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 "computedExpressionCompiledFunction.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.

6.77.1 Detailed Description

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

6.77.2 Macro Definition Documentation

6.77.2.1 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.
---	--

6.77.2.2 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.
---	---

6.78 src/program.cpp File Reference

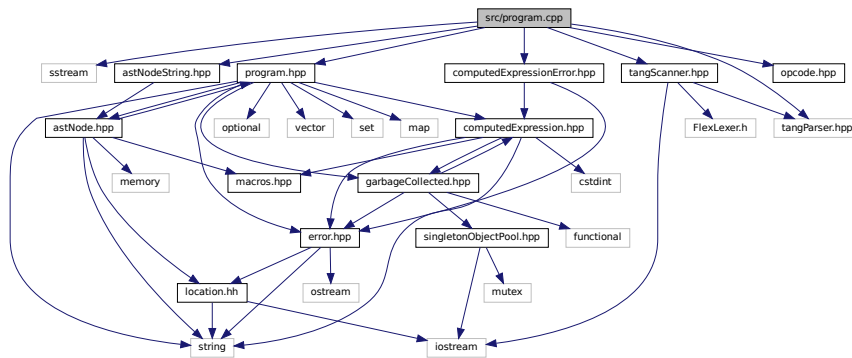
Define the [Tang::Program](#) class.


```

#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:



6.78.1 Detailed Description

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

6.79 src/tangBase.cpp File Reference

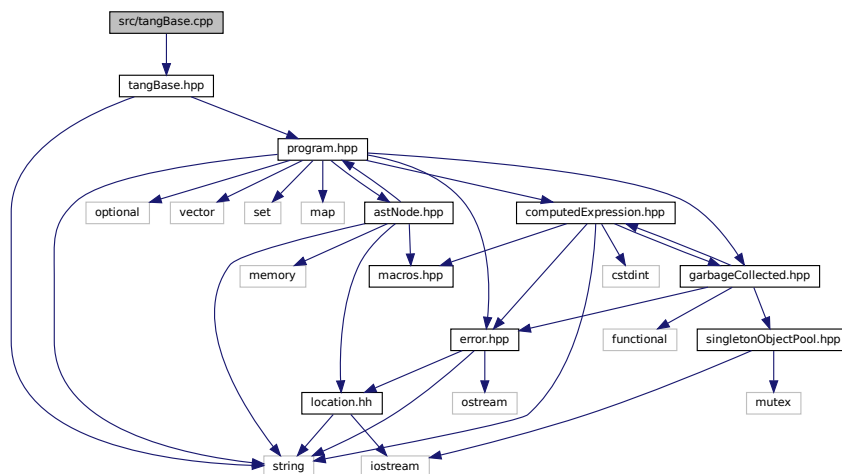
Define the [Tang::TangBase](#) class.

```

#include "tangBase.hpp"

```

Include dependency graph for tangBase.cpp:



6.79.1 Detailed Description

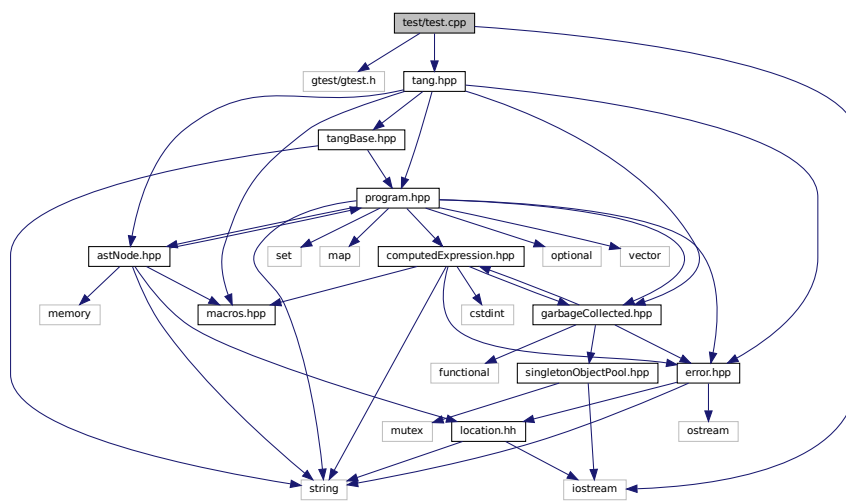
Define the [Tang::TangBase](#) class.

6.80 test/test.cpp File Reference

Test the general language behaviors.

```
#include <gtest/gtest.h>
#include <iostream>
#include "tang.hpp"
```

Include dependency graph for test.cpp:



Functions

- **TEST** (Declare, Null)
- **TEST** (Declare, Integer)
- **TEST** (Declare, Float)
- **TEST** (Declare, Boolean)
- **TEST** (Declare, String)
- **TEST** (Expression, Add)
- **TEST** (Expression, Subtract)
- **TEST** (Expression, Multiplication)
- **TEST** (Expression, Division)
- **TEST** (Expression, Modulo)
- **TEST** (Expression, UnaryMinus)
- **TEST** (Expression, Parentheses)
- **TEST** (Expression, TypeCast)
- **TEST** (Expression, Not)
- **TEST** (Expression, LessThan)
- **TEST** (Expression, LessThanEqual)
- **TEST** (Expression, GreaterThan)
- **TEST** (Expression, GreaterThanEqual)

- **TEST** (Expression, Equal)
- **TEST** (Expression, NotEqual)
- **TEST** (Expression, And)
- **TEST** (Expression, Or)
- **TEST** (Expression, Ternary)
- **TEST** (Expression, ArrayIndex)
- **TEST** (CodeBlock, Statements)
- **TEST** (Assign, Identifier)
- **TEST** (ControlFlow, IfElse)
- **TEST** (ControlFlow, While)
- **TEST** (ControlFlow, Break)
- **TEST** (ControlFlow, Continue)
- **TEST** (ControlFlow, DoWhile)
- **TEST** (ControlFlow, For)
- **TEST** (Print, Default)
- **TEST** (Function, Compiled)
- **TEST** (Function, Recursion)
- **TEST** (Function, FunctionCall)
- **TEST** (Function, Return)
- **int main** (int argc, char **argv)

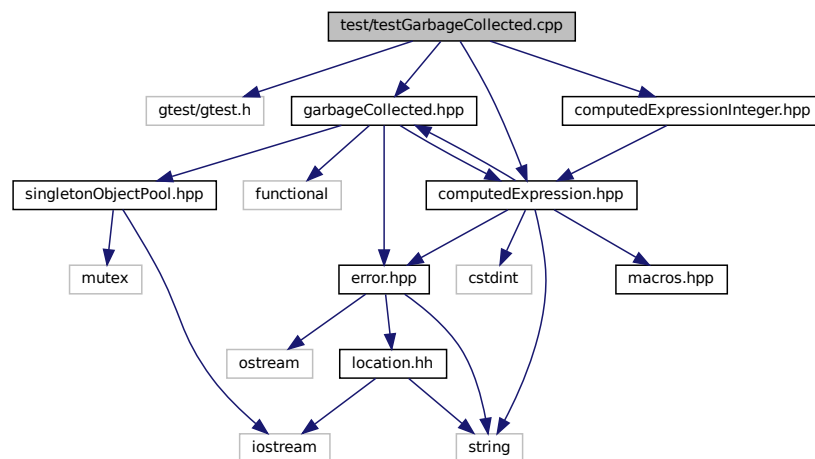
6.80.1 Detailed Description

Test the general language behaviors.

6.81 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)

6.81.1 Detailed Description

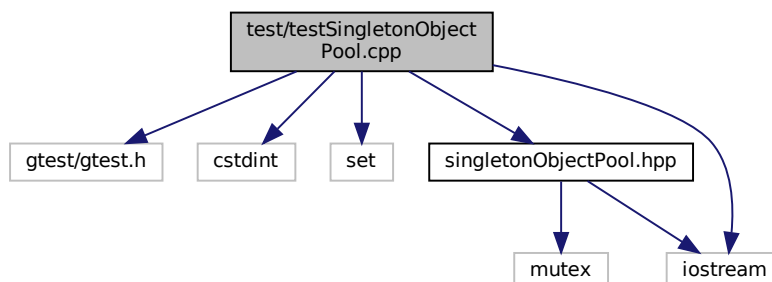
Test the generic behavior of the [Tang::GarbageCollected](#) class.

6.82 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)

6.82.1 Detailed Description

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

Index

- __add
 - Tang::ComputedExpression, [90](#)
 - Tang::ComputedExpressionArray, [100](#)
 - Tang::ComputedExpressionBoolean, [110](#)
 - Tang::ComputedExpressionCompiledFunction, [121](#)
 - Tang::ComputedExpressionError, [131](#)
 - Tang::ComputedExpressionFloat, [141](#)
 - Tang::ComputedExpressionInteger, [151](#)
 - Tang::ComputedExpressionString, [161](#)
- __boolean
 - Tang::ComputedExpression, [90](#)
 - Tang::ComputedExpressionArray, [100](#)
 - Tang::ComputedExpressionBoolean, [110](#)
 - Tang::ComputedExpressionCompiledFunction, [121](#)
 - Tang::ComputedExpressionError, [131](#)
 - Tang::ComputedExpressionFloat, [141](#)
 - Tang::ComputedExpressionInteger, [151](#)
 - Tang::ComputedExpressionString, [161](#)
- __divide
 - Tang::ComputedExpression, [90](#)
 - Tang::ComputedExpressionArray, [100](#)
 - Tang::ComputedExpressionBoolean, [110](#)
 - Tang::ComputedExpressionCompiledFunction, [121](#)
 - Tang::ComputedExpressionError, [131](#)
 - Tang::ComputedExpressionFloat, [141](#)
 - Tang::ComputedExpressionInteger, [151](#)
 - Tang::ComputedExpressionString, [161](#)
- __equal
 - Tang::ComputedExpression, [91](#)
 - Tang::ComputedExpressionArray, [101](#)
 - Tang::ComputedExpressionBoolean, [111](#)
 - Tang::ComputedExpressionCompiledFunction, [122](#)
 - Tang::ComputedExpressionError, [132](#)
 - Tang::ComputedExpressionFloat, [142](#)
 - Tang::ComputedExpressionInteger, [152](#)
 - Tang::ComputedExpressionString, [162](#)
- __float
 - Tang::ComputedExpression, [91](#)
 - Tang::ComputedExpressionArray, [101](#)
 - Tang::ComputedExpressionBoolean, [111](#)
 - Tang::ComputedExpressionCompiledFunction, [122](#)
 - Tang::ComputedExpressionError, [132](#)
 - Tang::ComputedExpressionFloat, [142](#)
 - Tang::ComputedExpressionInteger, [152](#)
 - Tang::ComputedExpressionString, [162](#)
- __index
 - Tang::ComputedExpression, [91](#)
 - Tang::ComputedExpressionArray, [101](#)
 - Tang::ComputedExpressionBoolean, [111](#)
 - Tang::ComputedExpressionCompiledFunction, [122](#)
 - Tang::ComputedExpressionError, [132](#)
 - Tang::ComputedExpressionFloat, [142](#)
 - Tang::ComputedExpressionInteger, [152](#)
 - Tang::ComputedExpressionString, [162](#)
- __integer
 - Tang::ComputedExpression, [92](#)
 - Tang::ComputedExpressionArray, [102](#)
 - Tang::ComputedExpressionBoolean, [112](#)
 - Tang::ComputedExpressionCompiledFunction, [123](#)
 - Tang::ComputedExpressionError, [133](#)
 - Tang::ComputedExpressionFloat, [143](#)
 - Tang::ComputedExpressionInteger, [153](#)
 - Tang::ComputedExpressionString, [163](#)
- __lessThan
 - Tang::ComputedExpression, [92](#)
 - Tang::ComputedExpressionArray, [102](#)
 - Tang::ComputedExpressionBoolean, [112](#)
 - Tang::ComputedExpressionCompiledFunction, [123](#)
 - Tang::ComputedExpressionError, [133](#)
 - Tang::ComputedExpressionFloat, [143](#)
 - Tang::ComputedExpressionInteger, [153](#)
 - Tang::ComputedExpressionString, [163](#)
- __modulo
 - Tang::ComputedExpression, [92](#)
 - Tang::ComputedExpressionArray, [102](#)
 - Tang::ComputedExpressionBoolean, [112](#)
 - Tang::ComputedExpressionCompiledFunction, [124](#)
 - Tang::ComputedExpressionError, [133](#)
 - Tang::ComputedExpressionFloat, [143](#)
 - Tang::ComputedExpressionInteger, [153](#)
 - Tang::ComputedExpressionString, [163](#)
- __multiply
 - Tang::ComputedExpression, [93](#)
 - Tang::ComputedExpressionArray, [103](#)
 - Tang::ComputedExpressionBoolean, [114](#)
 - Tang::ComputedExpressionCompiledFunction, [124](#)
 - Tang::ComputedExpressionError, [134](#)
 - Tang::ComputedExpressionFloat, [144](#)
 - Tang::ComputedExpressionInteger, [154](#)
 - Tang::ComputedExpressionString, [165](#)
- __negative
 - Tang::ComputedExpression, [93](#)
 - Tang::ComputedExpressionArray, [103](#)
 - Tang::ComputedExpressionBoolean, [114](#)
 - Tang::ComputedExpressionCompiledFunction, [124](#)
 - Tang::ComputedExpressionError, [134](#)
 - Tang::ComputedExpressionFloat, [144](#)
 - Tang::ComputedExpressionInteger, [154](#)

- Tang::ComputedExpressionString, 165
- __not
 - Tang::ComputedExpression, 93
 - Tang::ComputedExpressionArray, 103
 - Tang::ComputedExpressionBoolean, 114
 - Tang::ComputedExpressionCompiledFunction, 125
 - Tang::ComputedExpressionError, 134
 - Tang::ComputedExpressionFloat, 144
 - Tang::ComputedExpressionInteger, 154
 - Tang::ComputedExpressionString, 165
- __string
 - Tang::ComputedExpression, 94
 - Tang::ComputedExpressionArray, 104
 - Tang::ComputedExpressionBoolean, 115
 - Tang::ComputedExpressionCompiledFunction, 125
 - Tang::ComputedExpressionError, 135
 - Tang::ComputedExpressionFloat, 145
 - Tang::ComputedExpressionInteger, 155
 - Tang::ComputedExpressionString, 166
- __subtract
 - Tang::ComputedExpression, 94
 - Tang::ComputedExpressionArray, 104
 - Tang::ComputedExpressionBoolean, 115
 - Tang::ComputedExpressionCompiledFunction, 125
 - Tang::ComputedExpressionError, 135
 - Tang::ComputedExpressionFloat, 145
 - Tang::ComputedExpressionInteger, 155
 - Tang::ComputedExpressionString, 166
- ~GarbageCollected
 - Tang::GarbageCollected, 175
- ADD
 - opcode.hpp, 244
- Add
 - Tang::AstNodeBinary, 22
- addBreak
 - Tang::Program, 194
- addBytecode
 - Tang::Program, 194
- addContinue
 - Tang::Program, 194
- addIdentifier
 - Tang::Program, 194
- addString
 - Tang::Program, 195
- And
 - Tang::AstNodeBinary, 22
- ARRAY
 - opcode.hpp, 244
- AstNode
 - Tang::AstNode, 13
- AstNodeArray
 - Tang::AstNodeArray, 16
- AstNodeAssign
 - Tang::AstNodeAssign, 19
- AstNodeBinary
 - Tang::AstNodeBinary, 23
- AstNodeBlock
 - Tang::AstNodeBlock, 26
- AstNodeBoolean
 - Tang::AstNodeBoolean, 29
- AstNodeBreak
 - Tang::AstNodeBreak, 31
- AstNodeCast
 - Tang::AstNodeCast, 35
- AstNodeContinue
 - Tang::AstNodeContinue, 38
- AstNodeDoWhile
 - Tang::AstNodeDoWhile, 41
- AstNodeFloat
 - Tang::AstNodeFloat, 44
- AstNodeFor
 - Tang::AstNodeFor, 47
- AstNodeFunctionCall
 - Tang::AstNodeFunctionCall, 50
- AstNodeFunctionDeclaration
 - Tang::AstNodeFunctionDeclaration, 53
- AstNodeIdentifier
 - Tang::AstNodeIdentifier, 57
- AstNodeIfElse
 - Tang::AstNodeIfElse, 61
- AstNodeIndex
 - Tang::AstNodeIndex, 64
- AstNodeInteger
 - Tang::AstNodeInteger, 67
- AstNodePrint
 - Tang::AstNodePrint, 70
- AstNodeReturn
 - Tang::AstNodeReturn, 73
- AstNodeString
 - Tang::AstNodeString, 76
- AstNodeTernary
 - Tang::AstNodeTernary, 80
- AstNodeUnary
 - Tang::AstNodeUnary, 83
- AstNodeWhile
 - Tang::AstNodeWhile, 86
- BOOLEAN
 - opcode.hpp, 244
- Boolean
 - Tang::AstNodeCast, 35
- build/generated/location.hh, 207
- CALLFUNC
 - opcode.hpp, 244
- CASTBOOLEAN
 - opcode.hpp, 244
- CASTFLOAT
 - opcode.hpp, 244
- CASTINTEGER
 - opcode.hpp, 244
- CodeType
 - Tang::Program, 193
- compile
 - Tang::AstNode, 13
 - Tang::AstNodeArray, 17
 - Tang::AstNodeAssign, 19

- Tang::AstNodeBinary, [23](#)
- Tang::AstNodeBlock, [26](#)
- Tang::AstNodeBoolean, [29](#)
- Tang::AstNodeBreak, [32](#)
- Tang::AstNodeCast, [35](#)
- Tang::AstNodeContinue, [38](#)
- Tang::AstNodeDoWhile, [41](#)
- Tang::AstNodeFloat, [44](#)
- Tang::AstNodeFor, [47](#)
- Tang::AstNodeFunctionCall, [50](#)
- Tang::AstNodeFunctionDeclaration, [53](#)
- Tang::AstNodeIdentifier, [58](#)
- Tang::AstNodeIfElse, [61](#)
- Tang::AstNodeIndex, [64](#)
- Tang::AstNodeInteger, [67](#)
- Tang::AstNodePrint, [71](#)
- Tang::AstNodeReturn, [73](#)
- Tang::AstNodeString, [76](#)
- Tang::AstNodeTernary, [80](#)
- Tang::AstNodeUnary, [84](#)
- Tang::AstNodeWhile, [86](#)
- compileLiteral
 - Tang::AstNodeString, [77](#)
- compilePreprocess
 - Tang::AstNode, [14](#)
 - Tang::AstNodeArray, [17](#)
 - Tang::AstNodeAssign, [20](#)
 - Tang::AstNodeBinary, [24](#)
 - Tang::AstNodeBlock, [27](#)
 - Tang::AstNodeBoolean, [29](#)
 - Tang::AstNodeBreak, [32](#)
 - Tang::AstNodeCast, [36](#)
 - Tang::AstNodeContinue, [39](#)
 - Tang::AstNodeDoWhile, [42](#)
 - Tang::AstNodeFloat, [45](#)
 - Tang::AstNodeFor, [48](#)
 - Tang::AstNodeFunctionCall, [51](#)
 - Tang::AstNodeFunctionDeclaration, [54](#)
 - Tang::AstNodeIdentifier, [58](#)
 - Tang::AstNodeIfElse, [62](#)
 - Tang::AstNodeIndex, [65](#)
 - Tang::AstNodeInteger, [68](#)
 - Tang::AstNodePrint, [71](#)
 - Tang::AstNodeReturn, [74](#)
 - Tang::AstNodeString, [77](#)
 - Tang::AstNodeTernary, [81](#)
 - Tang::AstNodeUnary, [84](#)
 - Tang::AstNodeWhile, [87](#)
- compileScript
 - Tang::TangBase, [202](#)
- ComputedExpressionArray
 - Tang::ComputedExpressionArray, [99](#)
- ComputedExpressionBoolean
 - Tang::ComputedExpressionBoolean, [109](#)
- ComputedExpressionCompiledFunction
 - Tang::ComputedExpressionCompiledFunction, [120](#)
- ComputedExpressionError
 - Tang::ComputedExpressionError, [130](#)
- ComputedExpressionFloat
 - Tang::ComputedExpressionFloat, [140](#)
- ComputedExpressionInteger
 - Tang::ComputedExpressionInteger, [150](#)
- ComputedExpressionString
 - Tang::ComputedExpressionString, [160](#)
- Default
 - Tang::AstNodePrint, [70](#)
- DIVIDE
 - opcode.hpp, [244](#)
- Divide
 - Tang::AstNodeBinary, [22](#)
- dump
 - Tang::AstNode, [14](#)
 - Tang::AstNodeArray, [17](#)
 - Tang::AstNodeAssign, [20](#)
 - Tang::AstNodeBinary, [24](#)
 - Tang::AstNodeBlock, [27](#)
 - Tang::AstNodeBoolean, [30](#)
 - Tang::AstNodeBreak, [33](#)
 - Tang::AstNodeCast, [36](#)
 - Tang::AstNodeContinue, [39](#)
 - Tang::AstNodeDoWhile, [42](#)
 - Tang::AstNodeFloat, [45](#)
 - Tang::AstNodeFor, [48](#)
 - Tang::AstNodeFunctionCall, [51](#)
 - Tang::AstNodeFunctionDeclaration, [54](#)
 - Tang::AstNodeIdentifier, [59](#)
 - Tang::AstNodeIfElse, [62](#)
 - Tang::AstNodeIndex, [65](#)
 - Tang::AstNodeInteger, [68](#)
 - Tang::AstNodePrint, [71](#)
 - Tang::AstNodeReturn, [74](#)
 - Tang::AstNodeString, [78](#)
 - Tang::AstNodeTernary, [81](#)
 - Tang::AstNodeUnary, [84](#)
 - Tang::AstNodeWhile, [87](#)
 - Tang::ComputedExpression, [94](#)
 - Tang::ComputedExpressionArray, [104](#)
 - Tang::ComputedExpressionBoolean, [115](#)
 - Tang::ComputedExpressionCompiledFunction, [126](#)
 - Tang::ComputedExpressionError, [135](#)
 - Tang::ComputedExpressionFloat, [146](#)
 - Tang::ComputedExpressionInteger, [156](#)
 - Tang::ComputedExpressionString, [166](#)
- dumpBytecode
 - Tang::Program, [195](#)
- DUMPPROGRAMCHECK
 - program-dumpBytecode.cpp, [274](#)
- EQ
 - opcode.hpp, [244](#)
- Equal
 - Tang::AstNodeBinary, [22](#)
- Error
 - Tang::Error, [171](#)
- error.cpp
 - operator<<, [273](#)

- execute
 - Tang::Program, 195
- EXECUTEPROGRAMCHECK
 - program-execute.cpp, 276
- FLOAT
 - opcode.hpp, 244
- Float
 - Tang::AstNodeCast, 35
- FUNCTION
 - opcode.hpp, 244
- functionsDeclared
 - Tang::Program, 200
- GarbageCollected
 - Tang::GarbageCollected, 174, 175
- get
 - Tang::SingletonObjectPool< T >, 201
- get_next_token
 - Tang::TangScanner, 204
- getAst
 - Tang::Program, 196
- getBytecode
 - Tang::Program, 196
- getCode
 - Tang::Program, 196
- getIdentifiers
 - Tang::Program, 196
- getInstance
 - Tang::SingletonObjectPool< T >, 201
- getResult
 - Tang::Program, 197
- getStrings
 - Tang::Program, 197
- GreaterThan
 - Tang::AstNodeBinary, 22
- GreaterThanEqual
 - Tang::AstNodeBinary, 22
- GT
 - opcode.hpp, 244
- GTE
 - opcode.hpp, 244
- include/astNode.hpp, 209
- include/astNodeArray.hpp, 210
- include/astNodeAssign.hpp, 211
- include/astNodeBinary.hpp, 212
- include/astNodeBlock.hpp, 213
- include/astNodeBoolean.hpp, 214
- include/astNodeBreak.hpp, 215
- include/astNodeCast.hpp, 216
- include/astNodeContinue.hpp, 217
- include/astNodeDoWhile.hpp, 218
- include/astNodeFloat.hpp, 219
- include/astNodeFor.hpp, 220
- include/astNodeFunctionCall.hpp, 221
- include/astNodeFunctionDeclaration.hpp, 222
- include/astNodeIdentifier.hpp, 223
- include/astNodeIfElse.hpp, 224
- include/astNodeIndex.hpp, 225
- include/astNodeInteger.hpp, 226
- include/astNodePrint.hpp, 227
- include/astNodeReturn.hpp, 228
- include/astNodeString.hpp, 229
- include/astNodeTernary.hpp, 230
- include/astNodeUnary.hpp, 231
- include/astNodeWhile.hpp, 232
- include/computedExpression.hpp, 233
- include/computedExpressionArray.hpp, 234
- include/computedExpressionBoolean.hpp, 235
- include/computedExpressionCompiledFunction.hpp, 236
- include/computedExpressionError.hpp, 237
- include/computedExpressionFloat.hpp, 238
- include/computedExpressionInteger.hpp, 239
- include/computedExpressionString.hpp, 240
- include/error.hpp, 241
- include/garbageCollected.hpp, 242
- include/macros.hpp, 242
- include/opcode.hpp, 243
- include/program.hpp, 244
- include/singletonObjectPool.hpp, 245
- include/tang.hpp, 246
- include/tangBase.hpp, 247
- include/tangScanner.hpp, 249
- INDEX
 - opcode.hpp, 244
- INTEGER
 - opcode.hpp, 244
- Integer
 - Tang::AstNodeCast, 35
- is_equal
 - Tang::ComputedExpression, 95–97
 - Tang::ComputedExpressionArray, 105–107
 - Tang::ComputedExpressionBoolean, 116–118
 - Tang::ComputedExpressionCompiledFunction, 126–128
 - Tang::ComputedExpressionError, 136–138
 - Tang::ComputedExpressionFloat, 146–148
 - Tang::ComputedExpressionInteger, 156–158
 - Tang::ComputedExpressionString, 167–169
- JMP
 - opcode.hpp, 244
- JMPF
 - opcode.hpp, 244
- JMPF_POP
 - opcode.hpp, 244
- JMPT
 - opcode.hpp, 244
- JMPT_POP
 - opcode.hpp, 244
- LessThan
 - Tang::AstNodeBinary, 22
- LessThanEqual
 - Tang::AstNodeBinary, 22
- location.hh

- operator<<, [208](#), [209](#)
- LT
 - opcode.hpp, [244](#)
- LTE
 - opcode.hpp, [244](#)
- make
 - Tang::GarbageCollected, [175](#)
- makeCopy
 - Tang::ComputedExpression, [97](#)
 - Tang::ComputedExpressionArray, [107](#)
 - Tang::ComputedExpressionBoolean, [118](#)
 - Tang::ComputedExpressionCompiledFunction, [128](#)
 - Tang::ComputedExpressionError, [138](#)
 - Tang::ComputedExpressionFloat, [148](#)
 - Tang::ComputedExpressionInteger, [158](#)
 - Tang::ComputedExpressionString, [169](#)
- MODULO
 - opcode.hpp, [244](#)
- Modulo
 - Tang::AstNodeBinary, [22](#)
- MULTIPLY
 - opcode.hpp, [244](#)
- Multiply
 - Tang::AstNodeBinary, [22](#)
- NEGATIVE
 - opcode.hpp, [244](#)
- Negative
 - Tang::AstNodeUnary, [83](#)
- NEQ
 - opcode.hpp, [244](#)
- NOT
 - opcode.hpp, [244](#)
- Not
 - Tang::AstNodeUnary, [83](#)
- NotEqual
 - Tang::AstNodeBinary, [22](#)
- NULLVAL
 - opcode.hpp, [244](#)
- Opcode
 - opcode.hpp, [243](#)
- opcode.hpp
 - ADD, [244](#)
 - ARRAY, [244](#)
 - BOOLEAN, [244](#)
 - CALLFUNC, [244](#)
 - CASTBOOLEAN, [244](#)
 - CASTFLOAT, [244](#)
 - CASTINTEGER, [244](#)
 - DIVIDE, [244](#)
 - EQ, [244](#)
 - FLOAT, [244](#)
 - FUNCTION, [244](#)
 - GT, [244](#)
 - GTE, [244](#)
 - INDEX, [244](#)
 - INTEGER, [244](#)
 - JMP, [244](#)
 - JMPF, [244](#)
 - JMPF_POP, [244](#)
 - JMPT, [244](#)
 - JMPT_POP, [244](#)
 - LT, [244](#)
 - LTE, [244](#)
 - MODULO, [244](#)
 - MULTIPLY, [244](#)
 - NEGATIVE, [244](#)
 - NEQ, [244](#)
 - NOT, [244](#)
 - NULLVAL, [244](#)
 - Opcode, [243](#)
 - PEEK, [244](#)
 - POKE, [244](#)
 - POP, [244](#)
 - PRINT, [244](#)
 - RETURN, [244](#)
 - STRING, [244](#)
 - SUBTRACT, [244](#)
- Operation
 - Tang::AstNodeBinary, [22](#)
- Operator
 - Tang::AstNodeUnary, [83](#)
- operator!
 - Tang::GarbageCollected, [176](#)
- operator!=
 - Tang::GarbageCollected, [176](#)
- operator<
 - Tang::GarbageCollected, [181](#)
- operator<<
 - error.cpp, [273](#)
 - location.hh, [208](#), [209](#)
 - Tang::Error, [171](#)
 - Tang::GarbageCollected, [188](#)
- operator<=
 - Tang::GarbageCollected, [181](#)
- operator>
 - Tang::GarbageCollected, [187](#)
- operator>=
 - Tang::GarbageCollected, [187](#)
- operator*
 - Tang::GarbageCollected, [177](#), [178](#)
- operator+
 - Tang::GarbageCollected, [178](#)
- operator-
 - Tang::GarbageCollected, [179](#)
- operator->
 - Tang::GarbageCollected, [180](#)
- operator/
 - Tang::GarbageCollected, [180](#)
- operator=
 - Tang::GarbageCollected, [182](#)
- operator==
 - Tang::GarbageCollected, [183–185](#)
- operator%
 - Tang::GarbageCollected, [177](#)

- Or
 - Tang::AstNodeBinary, 22
- PEEK
 - opcode.hpp, 244
- POKE
 - opcode.hpp, 244
- POP
 - opcode.hpp, 244
- popBreakStack
 - Tang::Program, 197
- popContinueStack
 - Tang::Program, 198
- PRINT
 - opcode.hpp, 244
- Program
 - Tang::Program, 193
- program-dumpBytecode.cpp
 - DUMPPROGRAMCHECK, 274
- program-execute.cpp
 - EXECUTEPROGRAMCHECK, 276
 - STACKCHECK, 276
- pushEnvironment
 - Tang::Program, 198
- recycle
 - Tang::SingletonObjectPool< T >, 201
- RETURN
 - opcode.hpp, 244
- Script
 - Tang::Program, 193
- setFunctionStackDeclaration
 - Tang::Program, 199
- setJumpTarget
 - Tang::Program, 199
- src/astNode.cpp, 250
- src/astNodeArray.cpp, 250
- src/astNodeAssign.cpp, 251
- src/astNodeBinary.cpp, 252
- src/astNodeBlock.cpp, 253
- src/astNodeBoolean.cpp, 253
- src/astNodeBreak.cpp, 254
- src/astNodeCast.cpp, 255
- src/astNodeContinue.cpp, 255
- src/astNodeDoWhile.cpp, 256
- src/astNodeFloat.cpp, 257
- src/astNodeFor.cpp, 258
- src/astNodeFunctionCall.cpp, 258
- src/astNodeFunctionDeclaration.cpp, 259
- src/astNodeIdentifier.cpp, 260
- src/astNodeIfElse.cpp, 261
- src/astNodeIndex.cpp, 261
- src/astNodeInteger.cpp, 262
- src/astNodePrint.cpp, 263
- src/astNodeReturn.cpp, 263
- src/astNodeString.cpp, 264
- src/astNodeTernary.cpp, 265
- src/astNodeUnary.cpp, 266
- src/astNodeWhile.cpp, 266
- src/computedExpression.cpp, 267
- src/computedExpressionArray.cpp, 268
- src/computedExpressionBoolean.cpp, 269
- src/computedExpressionCompiledFunction.cpp, 269
- src/computedExpressionError.cpp, 270
- src/computedExpressionFloat.cpp, 271
- src/computedExpressionInteger.cpp, 271
- src/computedExpressionString.cpp, 272
- src/error.cpp, 273
- src/program-dumpBytecode.cpp, 274
- src/program-execute.cpp, 275
- src/program.cpp, 276
- src/tangBase.cpp, 277
- STACKCHECK
 - program-execute.cpp, 276
- STRING
 - opcode.hpp, 244
- SUBTRACT
 - opcode.hpp, 244
- Subtract
 - Tang::AstNodeBinary, 22
- Tang::AstNode, 11
 - AstNode, 13
 - compile, 13
 - compilePreprocess, 14
 - dump, 14
- Tang::AstNodeArray, 15
 - AstNodeArray, 16
 - compile, 17
 - compilePreprocess, 17
 - dump, 17
- Tang::AstNodeAssign, 18
 - AstNodeAssign, 19
 - compile, 19
 - compilePreprocess, 20
 - dump, 20
- Tang::AstNodeBinary, 21
 - Add, 22
 - And, 22
 - AstNodeBinary, 23
 - compile, 23
 - compilePreprocess, 24
 - Divide, 22
 - dump, 24
 - Equal, 22
 - GreaterThan, 22
 - GreaterThanEqual, 22
 - LessThan, 22
 - LessThanEqual, 22
 - Modulo, 22
 - Multiply, 22
 - NotEqual, 22
 - Operation, 22
 - Or, 22
 - Subtract, 22
- Tang::AstNodeBlock, 25
 - AstNodeBlock, 26

- compile, 26
- compilePreprocess, 27
- dump, 27
- Tang::AstNodeBoolean, 28
 - AstNodeBoolean, 29
 - compile, 29
 - compilePreprocess, 29
 - dump, 30
- Tang::AstNodeBreak, 30
 - AstNodeBreak, 31
 - compile, 32
 - compilePreprocess, 32
 - dump, 33
- Tang::AstNodeCast, 33
 - AstNodeCast, 35
 - Boolean, 35
 - compile, 35
 - compilePreprocess, 36
 - dump, 36
 - Float, 35
 - Integer, 35
 - Type, 35
- Tang::AstNodeContinue, 37
 - AstNodeContinue, 38
 - compile, 38
 - compilePreprocess, 39
 - dump, 39
- Tang::AstNodeDoWhile, 40
 - AstNodeDoWhile, 41
 - compile, 41
 - compilePreprocess, 42
 - dump, 42
- Tang::AstNodeFloat, 43
 - AstNodeFloat, 44
 - compile, 44
 - compilePreprocess, 45
 - dump, 45
- Tang::AstNodeFor, 46
 - AstNodeFor, 47
 - compile, 47
 - compilePreprocess, 48
 - dump, 48
- Tang::AstNodeFunctionCall, 49
 - AstNodeFunctionCall, 50
 - compile, 50
 - compilePreprocess, 51
 - dump, 51
- Tang::AstNodeFunctionDeclaration, 52
 - AstNodeFunctionDeclaration, 53
 - compile, 53
 - compilePreprocess, 54
 - dump, 54
- Tang::AstNodeIdentifier, 56
 - AstNodeIdentifier, 57
 - compile, 58
 - compilePreprocess, 58
 - dump, 59
- Tang::AstNodeIfElse, 59
 - AstNodeIfElse, 61
 - compile, 61
 - compilePreprocess, 62
 - dump, 62
- Tang::AstNodeIndex, 63
 - AstNodeIndex, 64
 - compile, 64
 - compilePreprocess, 65
 - dump, 65
- Tang::AstNodeInteger, 66
 - AstNodeInteger, 67
 - compile, 67
 - compilePreprocess, 68
 - dump, 68
- Tang::AstNodePrint, 69
 - AstNodePrint, 70
 - compile, 71
 - compilePreprocess, 71
 - Default, 70
 - dump, 71
 - Type, 70
- Tang::AstNodeReturn, 72
 - AstNodeReturn, 73
 - compile, 73
 - compilePreprocess, 74
 - dump, 74
- Tang::AstNodeString, 75
 - AstNodeString, 76
 - compile, 76
 - compileLiteral, 77
 - compilePreprocess, 77
 - dump, 78
- Tang::AstNodeTernary, 79
 - AstNodeTernary, 80
 - compile, 80
 - compilePreprocess, 81
 - dump, 81
- Tang::AstNodeUnary, 82
 - AstNodeUnary, 83
 - compile, 84
 - compilePreprocess, 84
 - dump, 84
 - Negative, 83
 - Not, 83
 - Operator, 83
- Tang::AstNodeWhile, 85
 - AstNodeWhile, 86
 - compile, 86
 - compilePreprocess, 87
 - dump, 87
- Tang::ComputedExpression, 88
 - __add, 90
 - __boolean, 90
 - __divide, 90
 - __equal, 91
 - __float, 91
 - __index, 91
 - __integer, 92

- __lessThan, 92
- __modulo, 92
- __multiply, 93
- __negative, 93
- __not, 93
- __string, 94
- __subtract, 94
- dump, 94
- is_equal, 95–97
- makeCopy, 97
- Tang::ComputedExpressionArray, 98
 - __add, 100
 - __boolean, 100
 - __divide, 100
 - __equal, 101
 - __float, 101
 - __index, 101
 - __integer, 102
 - __lessThan, 102
 - __modulo, 102
 - __multiply, 103
 - __negative, 103
 - __not, 103
 - __string, 104
 - __subtract, 104
 - ComputedExpressionArray, 99
 - dump, 104
 - is_equal, 105–107
 - makeCopy, 107
- Tang::ComputedExpressionBoolean, 108
 - __add, 110
 - __boolean, 110
 - __divide, 110
 - __equal, 111
 - __float, 111
 - __index, 111
 - __integer, 112
 - __lessThan, 112
 - __modulo, 112
 - __multiply, 114
 - __negative, 114
 - __not, 114
 - __string, 115
 - __subtract, 115
 - ComputedExpressionBoolean, 109
 - dump, 115
 - is_equal, 116–118
 - makeCopy, 118
- Tang::ComputedExpressionCompiledFunction, 119
 - __add, 121
 - __boolean, 121
 - __divide, 121
 - __equal, 122
 - __float, 122
 - __index, 122
 - __integer, 123
 - __lessThan, 123
 - __modulo, 124
 - __multiply, 124
 - __negative, 124
 - __not, 125
 - __string, 125
 - __subtract, 125
 - ComputedExpressionCompiledFunction, 120
 - dump, 126
 - is_equal, 126–128
 - makeCopy, 128
- Tang::ComputedExpressionError, 129
 - __add, 131
 - __boolean, 131
 - __divide, 131
 - __equal, 132
 - __float, 132
 - __index, 132
 - __integer, 133
 - __lessThan, 133
 - __modulo, 133
 - __multiply, 134
 - __negative, 134
 - __not, 134
 - __string, 135
 - __subtract, 135
 - ComputedExpressionError, 130
 - dump, 135
 - is_equal, 136–138
 - makeCopy, 138
- Tang::ComputedExpressionFloat, 139
 - __add, 141
 - __boolean, 141
 - __divide, 141
 - __equal, 142
 - __float, 142
 - __index, 142
 - __integer, 143
 - __lessThan, 143
 - __modulo, 143
 - __multiply, 144
 - __negative, 144
 - __not, 144
 - __string, 145
 - __subtract, 145
 - ComputedExpressionFloat, 140
 - dump, 146
 - is_equal, 146–148
 - makeCopy, 148
- Tang::ComputedExpressionInteger, 149
 - __add, 151
 - __boolean, 151
 - __divide, 151
 - __equal, 152
 - __float, 152
 - __index, 152
 - __integer, 153
 - __lessThan, 153
 - __modulo, 153
 - __multiply, 154

- __negative, 154
 - __not, 154
 - __string, 155
 - __subtract, 155
- ComputedExpressionInteger, 150
- dump, 156
- is_equal, 156–158
- makeCopy, 158
- Tang::ComputedExpressionString, 159
 - __add, 161
 - __boolean, 161
 - __divide, 161
 - __equal, 162
 - __float, 162
 - __index, 162
 - __integer, 163
 - __lessThan, 163
 - __modulo, 163
 - __multiply, 165
 - __negative, 165
 - __not, 165
 - __string, 166
 - __subtract, 166
- ComputedExpressionString, 160
- dump, 166
- is_equal, 167–169
- makeCopy, 169
- Tang::Error, 170
 - Error, 171
 - operator<<, 171
- Tang::GarbageCollected, 172
 - ~GarbageCollected, 175
- GarbageCollected, 174, 175
- make, 175
- operator!, 176
- operator!=, 176
- operator<, 181
- operator<<, 188
- operator<=, 181
- operator>, 187
- operator>=, 187
- operator*, 177, 178
- operator+, 178
- operator-, 179
- operator->, 180
- operator/, 180
- operator=, 182
- operator==, 183–185
- operator%, 177
- Tang::location, 188
- Tang::position, 190
- Tang::Program, 191
 - addBreak, 194
 - addBytecode, 194
 - addContinue, 194
 - addIdentifier, 194
 - addString, 195
- CodeType, 193
- dumpBytecode, 195
- execute, 195
- functionsDeclared, 200
- getAst, 196
- getBytecode, 196
- getCode, 196
- getIdentifiers, 196
- getResult, 197
- getStrings, 197
- popBreakStack, 197
- popContinueStack, 198
- Program, 193
- pushEnvironment, 198
- Script, 193
- setFunctionStackDeclaration, 199
- setJumpTarget, 199
- Template, 193
- Tang::SingletonObjectPool< T >, 200
 - get, 201
 - getInstance, 201
 - recycle, 201
- Tang::TangBase, 202
 - compileScript, 202
 - TangBase, 202
- Tang::TangScanner, 203
 - get_next_token, 204
 - TangScanner, 204
- TangBase
 - Tang::TangBase, 202
- TangScanner
 - Tang::TangScanner, 204
- Template
 - Tang::Program, 193
- test/test.cpp, 278
- test/testGarbageCollected.cpp, 279
- test/testSingletonObjectPool.cpp, 280
- Type
 - Tang::AstNodeCast, 35
 - Tang::AstNodePrint, 70