

Tang

0.1

Generated by Doxygen 1.9.1



<b>1 Tang: A Template Language</b>	<b>1</b>
1.1 Quick Description	1
1.2 Features	1
1.3 License	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Class Documentation</b>	<b>11</b>
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 collectIdentifiers()	13
5.2 Tang::AstNodeAssign Class Reference	14
5.2.1 Detailed Description	15
5.2.2 Constructor & Destructor Documentation	15
5.2.2.1 AstNodeAssign()	15
5.2.3 Member Function Documentation	15
5.2.3.1 collectIdentifiers()	15
5.3 Tang::AstNodeBinary Class Reference	16
5.3.1 Detailed Description	17
5.3.2 Member Enumeration Documentation	17
5.3.2.1 Operation	17
5.3.3 Constructor & Destructor Documentation	18
5.3.3.1 AstNodeBinary()	18
5.3.4 Member Function Documentation	18
5.3.4.1 collectIdentifiers()	18
5.4 Tang::AstNodeBlock Class Reference	18
5.4.1 Detailed Description	19
5.4.2 Constructor & Destructor Documentation	20
5.4.2.1 AstNodeBlock()	20
5.4.3 Member Function Documentation	20
5.4.3.1 collectIdentifiers()	20
5.5 Tang::AstNodeBoolean Class Reference	20
5.5.1 Detailed Description	21
5.5.2 Constructor & Destructor Documentation	21

5.5.2.1 AstNodeBoolean()	22
5.5.3 Member Function Documentation	22
5.5.3.1 collectIdentifiers()	22
5.6 Tang::AstNodeCast Class Reference	22
5.6.1 Detailed Description	23
5.6.2 Member Enumeration Documentation	24
5.6.2.1 Type	24
5.6.3 Constructor & Destructor Documentation	24
5.6.3.1 AstNodeCast()	24
5.6.4 Member Function Documentation	24
5.6.4.1 collectIdentifiers()	24
5.7 Tang::AstNodeDoWhile Class Reference	25
5.7.1 Detailed Description	26
5.7.2 Constructor & Destructor Documentation	26
5.7.2.1 AstNodeDoWhile()	26
5.7.3 Member Function Documentation	26
5.7.3.1 collectIdentifiers()	26
5.8 Tang::AstNodeFloat Class Reference	27
5.8.1 Detailed Description	28
5.8.2 Constructor & Destructor Documentation	28
5.8.2.1 AstNodeFloat()	28
5.8.3 Member Function Documentation	28
5.8.3.1 collectIdentifiers()	28
5.9 Tang::AstNodeFor Class Reference	29
5.9.1 Detailed Description	30
5.9.2 Constructor & Destructor Documentation	30
5.9.2.1 AstNodeFor()	30
5.9.3 Member Function Documentation	30
5.9.3.1 collectIdentifiers()	30
5.10 Tang::AstNodeIdentifier Class Reference	31
5.10.1 Detailed Description	32
5.10.2 Constructor & Destructor Documentation	32
5.10.2.1 AstNodeIdentifier()	32
5.10.3 Member Function Documentation	32
5.10.3.1 collectIdentifiers()	33
5.11 Tang::AstNodeIfElse Class Reference	33
5.11.1 Detailed Description	34
5.11.2 Constructor & Destructor Documentation	34
5.11.2.1 AstNodeIfElse() [1/2]	34
5.11.2.2 AstNodeIfElse() [2/2]	34
5.11.3 Member Function Documentation	36
5.11.3.1 collectIdentifiers()	36

5.12 Tang::AstNodeInteger Class Reference . . . . .	36
5.12.1 Detailed Description . . . . .	37
5.12.2 Constructor & Destructor Documentation . . . . .	37
5.12.2.1 AstNodeInteger() . . . . .	37
5.12.3 Member Function Documentation . . . . .	38
5.12.3.1 collectIdentifiers() . . . . .	38
5.13 Tang::AstNodePrint Class Reference . . . . .	38
5.13.1 Detailed Description . . . . .	39
5.13.2 Member Enumeration Documentation . . . . .	39
5.13.2.1 Type . . . . .	39
5.13.3 Constructor & Destructor Documentation . . . . .	40
5.13.3.1 AstNodePrint() . . . . .	40
5.13.4 Member Function Documentation . . . . .	40
5.13.4.1 collectIdentifiers() . . . . .	40
5.14 Tang::AstNodeString Class Reference . . . . .	41
5.14.1 Detailed Description . . . . .	41
5.14.2 Constructor & Destructor Documentation . . . . .	42
5.14.2.1 AstNodeString() . . . . .	42
5.14.3 Member Function Documentation . . . . .	42
5.14.3.1 collectIdentifiers() . . . . .	42
5.15 Tang::AstNodeUnary Class Reference . . . . .	42
5.15.1 Detailed Description . . . . .	43
5.15.2 Member Enumeration Documentation . . . . .	44
5.15.2.1 Operator . . . . .	44
5.15.3 Constructor & Destructor Documentation . . . . .	44
5.15.3.1 AstNodeUnary() . . . . .	44
5.15.4 Member Function Documentation . . . . .	44
5.15.4.1 collectIdentifiers() . . . . .	44
5.16 Tang::AstNodeWhile Class Reference . . . . .	45
5.16.1 Detailed Description . . . . .	46
5.16.2 Constructor & Destructor Documentation . . . . .	46
5.16.2.1 AstNodeWhile() . . . . .	46
5.16.3 Member Function Documentation . . . . .	46
5.16.3.1 collectIdentifiers() . . . . .	46
5.17 Tang::ComputedExpression Class Reference . . . . .	47
5.17.1 Detailed Description . . . . .	48
5.17.2 Member Function Documentation . . . . .	48
5.17.2.1 __add() . . . . .	48
5.17.2.2 __boolean() . . . . .	49
5.17.2.3 __divide() . . . . .	49
5.17.2.4 __equal() . . . . .	50
5.17.2.5 __float() . . . . .	50

5.17.2.6 <code>__integer()</code>	50
5.17.2.7 <code>__lessThan()</code>	50
5.17.2.8 <code>__modulo()</code>	51
5.17.2.9 <code>__multiply()</code>	51
5.17.2.10 <code>__negative()</code>	52
5.17.2.11 <code>__not()</code>	52
5.17.2.12 <code>__string()</code>	52
5.17.2.13 <code>__subtract()</code>	52
5.17.2.14 <code>dump()</code>	53
5.17.2.15 <code>is_equal()</code> [1/6]	53
5.17.2.16 <code>is_equal()</code> [2/6]	54
5.17.2.17 <code>is_equal()</code> [3/6]	55
5.17.2.18 <code>is_equal()</code> [4/6]	55
5.17.2.19 <code>is_equal()</code> [5/6]	56
5.17.2.20 <code>is_equal()</code> [6/6]	56
5.17.2.21 <code>makeCopy()</code>	56
5.18 Tang::ComputedExpressionBoolean Class Reference	57
5.18.1 Detailed Description	58
5.18.2 Constructor & Destructor Documentation	58
5.18.2.1 <code>ComputedExpressionBoolean()</code>	58
5.18.3 Member Function Documentation	59
5.18.3.1 <code>__add()</code>	59
5.18.3.2 <code>__boolean()</code>	59
5.18.3.3 <code>__divide()</code>	59
5.18.3.4 <code>__equal()</code>	60
5.18.3.5 <code>__float()</code>	60
5.18.3.6 <code>__integer()</code>	61
5.18.3.7 <code>__lessThan()</code>	61
5.18.3.8 <code>__modulo()</code>	61
5.18.3.9 <code>__multiply()</code>	62
5.18.3.10 <code>__negative()</code>	62
5.18.3.11 <code>__not()</code>	62
5.18.3.12 <code>__string()</code>	63
5.18.3.13 <code>__subtract()</code>	63
5.18.3.14 <code>dump()</code>	63
5.18.3.15 <code>is_equal()</code> [1/6]	63
5.18.3.16 <code>is_equal()</code> [2/6]	64
5.18.3.17 <code>is_equal()</code> [3/6]	64
5.18.3.18 <code>is_equal()</code> [4/6]	65
5.18.3.19 <code>is_equal()</code> [5/6]	65
5.18.3.20 <code>is_equal()</code> [6/6]	65
5.18.3.21 <code>makeCopy()</code>	66

5.19 Tang::ComputedExpressionError Class Reference	66
5.19.1 Detailed Description	68
5.19.2 Constructor & Destructor Documentation	68
5.19.2.1 ComputedExpressionError()	68
5.19.3 Member Function Documentation	68
5.19.3.1 __add()	68
5.19.3.2 __boolean()	69
5.19.3.3 __divide()	69
5.19.3.4 __equal()	69
5.19.3.5 __float()	70
5.19.3.6 __integer()	70
5.19.3.7 __lessThan()	70
5.19.3.8 __modulo()	71
5.19.3.9 __multiply()	71
5.19.3.10 __negative()	72
5.19.3.11 __not()	72
5.19.3.12 __string()	72
5.19.3.13 __subtract()	72
5.19.3.14 dump()	73
5.19.3.15 is_equal() [1/6]	73
5.19.3.16 is_equal() [2/6]	73
5.19.3.17 is_equal() [3/6]	74
5.19.3.18 is_equal() [4/6]	74
5.19.3.19 is_equal() [5/6]	75
5.19.3.20 is_equal() [6/6]	75
5.19.3.21 makeCopy()	75
5.20 Tang::ComputedExpressionFloat Class Reference	76
5.20.1 Detailed Description	77
5.20.2 Constructor & Destructor Documentation	77
5.20.2.1 ComputedExpressionFloat()	77
5.20.3 Member Function Documentation	78
5.20.3.1 __add()	78
5.20.3.2 __boolean()	78
5.20.3.3 __divide()	78
5.20.3.4 __equal()	79
5.20.3.5 __float()	79
5.20.3.6 __integer()	80
5.20.3.7 __lessThan()	80
5.20.3.8 __modulo()	80
5.20.3.9 __multiply()	81
5.20.3.10 __negative()	81
5.20.3.11 __not()	81

5.20.3.12	__string()	82
5.20.3.13	__subtract()	82
5.20.3.14	dump()	82
5.20.3.15	is_equal() [1/6]	83
5.20.3.16	is_equal() [2/6]	83
5.20.3.17	is_equal() [3/6]	83
5.20.3.18	is_equal() [4/6]	84
5.20.3.19	is_equal() [5/6]	84
5.20.3.20	is_equal() [6/6]	85
5.20.3.21	makeCopy()	85
5.21	Tang::ComputedExpressionInteger Class Reference	85
5.21.1	Detailed Description	87
5.21.2	Constructor & Destructor Documentation	87
5.21.2.1	ComputedExpressionInteger()	87
5.21.3	Member Function Documentation	88
5.21.3.1	__add()	88
5.21.3.2	__boolean()	88
5.21.3.3	__divide()	88
5.21.3.4	__equal()	89
5.21.3.5	__float()	89
5.21.3.6	__integer()	90
5.21.3.7	__lessThan()	90
5.21.3.8	__modulo()	90
5.21.3.9	__multiply()	91
5.21.3.10	__negative()	91
5.21.3.11	__not()	91
5.21.3.12	__string()	92
5.21.3.13	__subtract()	92
5.21.3.14	dump()	92
5.21.3.15	is_equal() [1/6]	93
5.21.3.16	is_equal() [2/6]	93
5.21.3.17	is_equal() [3/6]	93
5.21.3.18	is_equal() [4/6]	94
5.21.3.19	is_equal() [5/6]	94
5.21.3.20	is_equal() [6/6]	95
5.21.3.21	makeCopy()	95
5.22	Tang::ComputedExpressionString Class Reference	95
5.22.1	Detailed Description	97
5.22.2	Constructor & Destructor Documentation	97
5.22.2.1	ComputedExpressionString()	97
5.22.3	Member Function Documentation	98
5.22.3.1	__add()	98



5.22.3.2 <code>__boolean()</code>	98
5.22.3.3 <code>__divide()</code>	98
5.22.3.4 <code>__equal()</code>	99
5.22.3.5 <code>__float()</code>	99
5.22.3.6 <code>__integer()</code>	99
5.22.3.7 <code>__lessThan()</code>	100
5.22.3.8 <code>__modulo()</code>	100
5.22.3.9 <code>__multiply()</code>	100
5.22.3.10 <code>__negative()</code>	101
5.22.3.11 <code>__not()</code>	101
5.22.3.12 <code>__string()</code>	101
5.22.3.13 <code>__subtract()</code>	101
5.22.3.14 <code>dump()</code>	102
5.22.3.15 <code>is_equal()</code> [1/6]	102
5.22.3.16 <code>is_equal()</code> [2/6]	102
5.22.3.17 <code>is_equal()</code> [3/6]	103
5.22.3.18 <code>is_equal()</code> [4/6]	103
5.22.3.19 <code>is_equal()</code> [5/6]	104
5.22.3.20 <code>is_equal()</code> [6/6]	104
5.22.3.21 <code>makeCopy()</code>	104
5.23 Tang::Error Class Reference	105
5.23.1 Detailed Description	106
5.23.2 Constructor & Destructor Documentation	106
5.23.2.1 <code>Error()</code> [1/2]	106
5.23.2.2 <code>Error()</code> [2/2]	106
5.23.3 Friends And Related Function Documentation	106
5.23.3.1 <code>operator&lt;&lt;</code>	107
5.24 Tang::GarbageCollected Class Reference	107
5.24.1 Detailed Description	109
5.24.2 Constructor & Destructor Documentation	109
5.24.2.1 <code>GarbageCollected()</code> [1/3]	109
5.24.2.2 <code>GarbageCollected()</code> [2/3]	110
5.24.2.3 <code>~GarbageCollected()</code>	110
5.24.2.4 <code>GarbageCollected()</code> [3/3]	110
5.24.3 Member Function Documentation	110
5.24.3.1 <code>make()</code>	110
5.24.3.2 <code>operator"()!</code>	111
5.24.3.3 <code>operator"!=(</code>	111
5.24.3.4 <code>operator%(</code>	112
5.24.3.5 <code>operator*(</code> [1/2]	113
5.24.3.6 <code>operator*(</code> [2/2]	113
5.24.3.7 <code>operator+(</code>	113

5.24.3.8 operator-() [1/2]	114
5.24.3.9 operator-() [2/2]	114
5.24.3.10 operator->()	115
5.24.3.11 operator/()	115
5.24.3.12 operator<()	116
5.24.3.13 operator<=()	116
5.24.3.14 operator=() [1/2]	117
5.24.3.15 operator=() [2/2]	117
5.24.3.16 operator==( ) [1/8]	118
5.24.3.17 operator==( ) [2/8]	118
5.24.3.18 operator==( ) [3/8]	119
5.24.3.19 operator==( ) [4/8]	119
5.24.3.20 operator==( ) [5/8]	119
5.24.3.21 operator==( ) [6/8]	120
5.24.3.22 operator==( ) [7/8]	120
5.24.3.23 operator==( ) [8/8]	121
5.24.3.24 operator>()	121
5.24.3.25 operator>=()	121
5.24.4 Friends And Related Function Documentation	122
5.24.4.1 operator<<	122
5.25 Tang::location Class Reference	122
5.25.1 Detailed Description	124
5.26 Tang::position Class Reference	124
5.26.1 Detailed Description	125
5.27 Tang::Program Class Reference	125
5.27.1 Detailed Description	127
5.27.2 Member Enumeration Documentation	127
5.27.2.1 CodeType	127
5.27.3 Constructor & Destructor Documentation	127
5.27.3.1 Program()	127
5.27.4 Member Function Documentation	128
5.27.4.1 addBytecode()	128
5.27.4.2 dumpBytecode()	128
5.27.4.3 execute()	128
5.27.4.4 getAst()	129
5.27.4.5 getBytecode()	129
5.27.4.6 getCode()	129
5.27.4.7 getResult()	129
5.27.4.8 setJumpTarget()	129
5.28 Tang::SingletonObjectPool< T > Class Template Reference	130
5.28.1 Detailed Description	130
5.28.2 Member Function Documentation	130

5.28.2.1 get()	131
5.28.2.2 getInstance()	131
5.28.2.3 recycle()	131
5.29 Tang::TangBase Class Reference	131
5.29.1 Detailed Description	132
5.29.2 Constructor & Destructor Documentation	132
5.29.2.1 TangBase()	132
5.29.3 Member Function Documentation	132
5.29.3.1 compileScript()	132
5.30 Tang::TangScanner Class Reference	133
5.30.1 Detailed Description	134
5.30.2 Constructor & Destructor Documentation	134
5.30.2.1 TangScanner()	134
5.30.3 Member Function Documentation	134
5.30.3.1 get_next_token()	134
<b>6 File Documentation</b>	<b>135</b>
6.1 build/generated/location.hh File Reference	135
6.1.1 Detailed Description	136
6.1.2 Function Documentation	136
6.1.2.1 operator<<() [1/2]	136
6.1.2.2 operator<<() [2/2]	137
6.2 include/astNode.hpp File Reference	137
6.2.1 Detailed Description	138
6.3 include/astNodeAssign.hpp File Reference	138
6.3.1 Detailed Description	139
6.4 include/astNodeBinary.hpp File Reference	139
6.4.1 Detailed Description	140
6.5 include/astNodeBlock.hpp File Reference	140
6.5.1 Detailed Description	141
6.6 include/astNodeBoolean.hpp File Reference	141
6.6.1 Detailed Description	142
6.7 include/astNodeCast.hpp File Reference	142
6.7.1 Detailed Description	143
6.8 include/astNodeDoWhile.hpp File Reference	143
6.8.1 Detailed Description	144
6.9 include/astNodeFloat.hpp File Reference	144
6.9.1 Detailed Description	145
6.10 include/astNodeFor.hpp File Reference	145
6.10.1 Detailed Description	146
6.11 include/astNodeIdentifier.hpp File Reference	146
6.11.1 Detailed Description	147

6.12 include/astNodeIfElse.hpp File Reference	147
6.12.1 Detailed Description	148
6.13 include/astNodeInteger.hpp File Reference	148
6.13.1 Detailed Description	149
6.14 include/astNodePrint.hpp File Reference	149
6.14.1 Detailed Description	150
6.15 include/astNodeString.hpp File Reference	150
6.15.1 Detailed Description	151
6.16 include/astNodeUnary.hpp File Reference	151
6.16.1 Detailed Description	152
6.17 include/astNodeWhile.hpp File Reference	152
6.17.1 Detailed Description	153
6.18 include/computedExpression.hpp File Reference	153
6.18.1 Detailed Description	154
6.19 include/computedExpressionBoolean.hpp File Reference	154
6.19.1 Detailed Description	155
6.20 include/computedExpressionError.hpp File Reference	155
6.20.1 Detailed Description	156
6.21 include/computedExpressionFloat.hpp File Reference	156
6.21.1 Detailed Description	156
6.22 include/computedExpressionInteger.hpp File Reference	157
6.22.1 Detailed Description	157
6.23 include/computedExpressionString.hpp File Reference	158
6.23.1 Detailed Description	158
6.24 include/error.hpp File Reference	159
6.24.1 Detailed Description	159
6.25 include/garbageCollected.hpp File Reference	160
6.25.1 Detailed Description	160
6.26 include/macros.hpp File Reference	160
6.26.1 Detailed Description	161
6.26.2 Macro Definition Documentation	161
6.26.2.1 TANG_UNUSED	161
6.27 include/opcode.hpp File Reference	161
6.27.1 Detailed Description	162
6.27.2 Enumeration Type Documentation	162
6.27.2.1 Opcode	162
6.28 include/program.hpp File Reference	163
6.28.1 Detailed Description	163
6.29 include/singletonObjectPool.hpp File Reference	164
6.29.1 Detailed Description	164
6.30 include/tang.hpp File Reference	165
6.30.1 Detailed Description	165

6.31 include/tangBase.hpp File Reference . . . . .	166
6.31.1 Detailed Description . . . . .	167
6.32 include/tangScanner.hpp File Reference . . . . .	167
6.32.1 Detailed Description . . . . .	168
6.33 src/astNode.cpp File Reference . . . . .	168
6.33.1 Detailed Description . . . . .	168
6.34 src/astNodeAssign.cpp File Reference . . . . .	168
6.34.1 Detailed Description . . . . .	169
6.35 src/astNodeBinary.cpp File Reference . . . . .	169
6.35.1 Detailed Description . . . . .	170
6.36 src/astNodeBlock.cpp File Reference . . . . .	170
6.36.1 Detailed Description . . . . .	170
6.37 src/astNodeBoolean.cpp File Reference . . . . .	170
6.37.1 Detailed Description . . . . .	171
6.38 src/astNodeCast.cpp File Reference . . . . .	171
6.38.1 Detailed Description . . . . .	172
6.39 src/astNodeDoWhile.cpp File Reference . . . . .	172
6.39.1 Detailed Description . . . . .	173
6.40 src/astNodeFloat.cpp File Reference . . . . .	173
6.40.1 Detailed Description . . . . .	174
6.41 src/astNodeFor.cpp File Reference . . . . .	174
6.41.1 Detailed Description . . . . .	174
6.42 src/astNodeIdentifier.cpp File Reference . . . . .	174
6.42.1 Detailed Description . . . . .	175
6.43 src/astNodeIfElse.cpp File Reference . . . . .	175
6.43.1 Detailed Description . . . . .	176
6.44 src/astNodeInteger.cpp File Reference . . . . .	176
6.44.1 Detailed Description . . . . .	177
6.45 src/astNodePrint.cpp File Reference . . . . .	177
6.45.1 Detailed Description . . . . .	177
6.46 src/astNodeString.cpp File Reference . . . . .	177
6.46.1 Detailed Description . . . . .	178
6.47 src/astNodeUnary.cpp File Reference . . . . .	178
6.47.1 Detailed Description . . . . .	179
6.48 src/astNodeWhile.cpp File Reference . . . . .	179
6.48.1 Detailed Description . . . . .	180
6.49 src/computedExpression.cpp File Reference . . . . .	180
6.49.1 Detailed Description . . . . .	180
6.50 src/computedExpressionBoolean.cpp File Reference . . . . .	180
6.50.1 Detailed Description . . . . .	181
6.51 src/computedExpressionError.cpp File Reference . . . . .	181
6.51.1 Detailed Description . . . . .	182

6.52 src/computedExpressionFloat.cpp File Reference . . . . .	182
6.52.1 Detailed Description . . . . .	182
6.53 src/computedExpressionInteger.cpp File Reference . . . . .	182
6.53.1 Detailed Description . . . . .	183
6.54 src/computedExpressionString.cpp File Reference . . . . .	183
6.54.1 Detailed Description . . . . .	184
6.55 src/error.cpp File Reference . . . . .	184
6.55.1 Detailed Description . . . . .	184
6.55.2 Function Documentation . . . . .	184
6.55.2.1 operator<<() . . . . .	184
6.56 src/program-dumpBytecode.cpp File Reference . . . . .	185
6.56.1 Detailed Description . . . . .	185
6.56.2 Macro Definition Documentation . . . . .	186
6.56.2.1 DUMPPROGRAMCHECK . . . . .	186
6.57 src/program-execute.cpp File Reference . . . . .	186
6.57.1 Detailed Description . . . . .	187
6.57.2 Macro Definition Documentation . . . . .	187
6.57.2.1 EXECUTEPROGRAMCHECK . . . . .	187
6.57.2.2 STACKCHECK . . . . .	187
6.58 src/program.cpp File Reference . . . . .	188
6.58.1 Detailed Description . . . . .	188
6.59 src/tangBase.cpp File Reference . . . . .	188
6.59.1 Detailed Description . . . . .	189
6.60 test/test.cpp File Reference . . . . .	189
6.60.1 Detailed Description . . . . .	190
6.61 test/testGarbageCollected.cpp File Reference . . . . .	190
6.61.1 Detailed Description . . . . .	191
6.62 test/testSingletonObjectPool.cpp File Reference . . . . .	191
6.62.1 Detailed Description . . . . .	192

# 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::AstNodeAssign . . . . .	14
Tang::AstNodeBinary . . . . .	16
Tang::AstNodeBlock . . . . .	18
Tang::AstNodeBoolean . . . . .	20
Tang::AstNodeCast . . . . .	22
Tang::AstNodeDoWhile . . . . .	25
Tang::AstNodeFloat . . . . .	27
Tang::AstNodeFor . . . . .	29
Tang::AstNodeIdentifier . . . . .	31
Tang::AstNodeIfElse . . . . .	33
Tang::AstNodeInteger . . . . .	36
Tang::AstNodePrint . . . . .	38
Tang::AstNodeString . . . . .	41
Tang::AstNodeUnary . . . . .	42
Tang::AstNodeWhile . . . . .	45
Tang::ComputedExpression . . . . .	47
Tang::ComputedExpressionBoolean . . . . .	57
Tang::ComputedExpressionError . . . . .	66
Tang::ComputedExpressionFloat . . . . .	76
Tang::ComputedExpressionInteger . . . . .	85
Tang::ComputedExpressionString . . . . .	95
Tang::Error . . . . .	105
Tang::GarbageCollected . . . . .	107
Tang::location . . . . .	122
Tang::position . . . . .	124
Tang::Program . . . . .	125
Tang::SingletonObjectPool< T > . . . . .	130
Tang::TangBase . . . . .	131
TangTangFlexLexer	
Tang::TangScanner . . . . .	133



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Tang::AstNode</a>	Base class for representing nodes of an Abstract Syntax Tree (AST) . . . . .	11
<a href="#">Tang::AstNodeAssign</a>	An <a href="#">AstNode</a> that represents a binary expression . . . . .	14
<a href="#">Tang::AstNodeBinary</a>	An <a href="#">AstNode</a> that represents a binary expression . . . . .	16
<a href="#">Tang::AstNodeBlock</a>	An <a href="#">AstNode</a> that represents a code block . . . . .	18
<a href="#">Tang::AstNodeBoolean</a>	An <a href="#">AstNode</a> that represents a boolean literal . . . . .	20
<a href="#">Tang::AstNodeCast</a>	An <a href="#">AstNode</a> that represents a typecast of an expression . . . . .	22
<a href="#">Tang::AstNodeDoWhile</a>	An <a href="#">AstNode</a> that represents a do..while statement . . . . .	25
<a href="#">Tang::AstNodeFloat</a>	An <a href="#">AstNode</a> that represents an float literal . . . . .	27
<a href="#">Tang::AstNodeFor</a>	An <a href="#">AstNode</a> that represents an if() statement . . . . .	29
<a href="#">Tang::AstNodeIdentifier</a>	An <a href="#">AstNode</a> that represents an identifier . . . . .	31
<a href="#">Tang::AstNodeIfElse</a>	An <a href="#">AstNode</a> that represents an if..else statement . . . . .	33
<a href="#">Tang::AstNodeInteger</a>	An <a href="#">AstNode</a> that represents an integer literal . . . . .	36
<a href="#">Tang::AstNodePrint</a>	An <a href="#">AstNode</a> that represents a print typeoperation . . . . .	38
<a href="#">Tang::AstNodeString</a>	An <a href="#">AstNode</a> that represents a string literal . . . . .	41
<a href="#">Tang::AstNodeUnary</a>	An <a href="#">AstNode</a> that represents a unary negation . . . . .	42
<a href="#">Tang::AstNodeWhile</a>	An <a href="#">AstNode</a> that represents a while statement . . . . .	45
<a href="#">Tang::ComputedExpression</a>	Represents the result of a computation that has been executed . . . . .	47
<a href="#">Tang::ComputedExpressionBoolean</a>	Represents an Boolean that is the result of a computation . . . . .	57

<a href="#">Tang::ComputedExpressionError</a>	
Represents a Runtime <a href="#">Error</a> . . . . .	66
<a href="#">Tang::ComputedExpressionFloat</a>	
Represents a Float that is the result of a computation . . . . .	76
<a href="#">Tang::ComputedExpressionInteger</a>	
Represents an Integer that is the result of a computation . . . . .	85
<a href="#">Tang::ComputedExpressionString</a>	
Represents a String that is the result of a computation . . . . .	95
<a href="#">Tang::Error</a>	
Used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error . . . . .	105
<a href="#">Tang::GarbageCollected</a>	
A container that acts as a resource-counting garbage collector for the specified type . . . . .	107
<a href="#">Tang::location</a>	
Two points in a source file . . . . .	122
<a href="#">Tang::position</a>	
A point in a source file . . . . .	124
<a href="#">Tang::Program</a>	
Represents a compiled script or template that may be executed . . . . .	125
<a href="#">Tang::SingletonObjectPool&lt; T &gt;</a>	
A thread-safe, singleton object pool of the designated type . . . . .	130
<a href="#">Tang::TangBase</a>	
The base class for the Tang programming language . . . . .	131
<a href="#">Tang::TangScanner</a>	
The Flex lexer class for the main Tang language . . . . .	133

## 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 . . . . .	135
include/astNode.hpp	
Declare the <code>Tang::AstNode</code> base class . . . . .	137
include/astNodeAssign.hpp	
Declare the <code>Tang::AstNodeAssign</code> class . . . . .	138
include/astNodeBinary.hpp	
Declare the <code>Tang::AstNodeBinary</code> class . . . . .	139
include/astNodeBlock.hpp	
Declare the <code>Tang::AstNodeBlock</code> class . . . . .	140
include/astNodeBoolean.hpp	
Declare the <code>Tang::AstNodeBoolean</code> class . . . . .	141
include/astNodeCast.hpp	
Declare the <code>Tang::AstNodeCast</code> class . . . . .	142
include/astNodeDoWhile.hpp	
Declare the <code>Tang::AstNodeDoWhile</code> class . . . . .	143
include/astNodeFloat.hpp	
Declare the <code>Tang::AstNodeFloat</code> class . . . . .	144
include/astNodeFor.hpp	
Declare the <code>Tang::AstNodeFor</code> class . . . . .	145
include/astNodeIdentifier.hpp	
Declare the <code>Tang::AstNodeIdentifier</code> class . . . . .	146
include/astNodeIfElse.hpp	
Declare the <code>Tang::AstNodeIfElse</code> class . . . . .	147
include/astNodeInteger.hpp	
Declare the <code>Tang::AstNodeInteger</code> class . . . . .	148
include/astNodePrint.hpp	
Declare the <code>Tang::AstNodePrint</code> class . . . . .	149
include/astNodeString.hpp	
Declare the <code>Tang::AstNodeString</code> class . . . . .	150
include/astNodeUnary.hpp	
Declare the <code>Tang::AstNodeUnary</code> class . . . . .	151
include/astNodeWhile.hpp	
Declare the <code>Tang::AstNodeWhile</code> class . . . . .	152
include/computedExpression.hpp	
Declare the <code>Tang::ComputedExpression</code> base class . . . . .	153

include/computedExpressionBoolean.hpp	
Declare the <a href="#">Tang::ComputedExpressionBoolean</a> class	154
include/computedExpressionError.hpp	
Declare the <a href="#">Tang::ComputedExpressionError</a> class	155
include/computedExpressionFloat.hpp	
Declare the <a href="#">Tang::ComputedExpressionFloat</a> class	156
include/computedExpressionInteger.hpp	
Declare the <a href="#">Tang::ComputedExpressionInteger</a> class	157
include/computedExpressionString.hpp	
Declare the <a href="#">Tang::ComputedExpressionString</a> class	158
include/error.hpp	
Declare the <a href="#">Tang::Error</a> class used to describe syntax and runtime errors	159
include/garbageCollected.hpp	
Declare the <a href="#">Tang::GarbageCollected</a> class	160
include/macros.hpp	
Contains generic macros	160
include/opcode.hpp	
Declare the Opcodes used in the Bytecode representation of a program	161
include/program.hpp	
Declare the <a href="#">Tang::Program</a> class used to compile and execute source code	163
include/singletonObjectPool.hpp	
Declare the <a href="#">Tang::SingletonObjectPool</a> class	164
include/tang.hpp	
Header file supplied for use by 3rd party code so that they can easily include all necessary headers	165
include/tangBase.hpp	
Declare the <a href="#">Tang::TangBase</a> class used to interact with Tang	166
include/tangScanner.hpp	
Declare the <a href="#">Tang::TangScanner</a> used to tokenize a Tang script	167
src/astNode.cpp	
Define the <a href="#">Tang::AstNode</a> class	168
src/astNodeAssign.cpp	
Define the <a href="#">Tang::AstNodeAssign</a> class	168
src/astNodeBinary.cpp	
Define the <a href="#">Tang::AstNodeBinary</a> class	169
src/astNodeBlock.cpp	
Define the <a href="#">Tang::AstNodeBlock</a> class	170
src/astNodeBoolean.cpp	
Define the <a href="#">Tang::AstNodeBoolean</a> class	170
src/astNodeCast.cpp	
Define the <a href="#">Tang::AstNodeCast</a> class	171
src/astNodeDoWhile.cpp	
Define the <a href="#">Tang::AstNodeDoWhile</a> class	172
src/astNodeFloat.cpp	
Define the <a href="#">Tang::AstNodeFloat</a> class	173
src/astNodeFor.cpp	
Define the <a href="#">Tang::AstNodeFor</a> class	174
src/astNodeIdentifier.cpp	
Define the <a href="#">Tang::AstNodeIdentifier</a> class	174
src/astNodeIfElse.cpp	
Define the <a href="#">Tang::AstNodeIfElse</a> class	175
src/astNodeInteger.cpp	
Define the <a href="#">Tang::AstNodeInteger</a> class	176
src/astNodePrint.cpp	
Define the <a href="#">Tang::AstNodePrint</a> class	177
src/astNodeString.cpp	
Define the <a href="#">Tang::AstNodeString</a> class	177

src/ <a href="#">astNodeUnary.cpp</a>	
Define the <a href="#">Tang::AstNodeUnary</a> class	178
src/ <a href="#">astNodeWhile.cpp</a>	
Define the <a href="#">Tang::AstNodeWhile</a> class	179
src/ <a href="#">computedExpression.cpp</a>	
Define the <a href="#">Tang::ComputedExpression</a> class	180
src/ <a href="#">computedExpressionBoolean.cpp</a>	
Define the <a href="#">Tang::ComputedExpressionBoolean</a> class	180
src/ <a href="#">computedExpressionError.cpp</a>	
Define the <a href="#">Tang::ComputedExpressionError</a> class	181
src/ <a href="#">computedExpressionFloat.cpp</a>	
Define the <a href="#">Tang::ComputedExpressionFloat</a> class	182
src/ <a href="#">computedExpressionInteger.cpp</a>	
Define the <a href="#">Tang::ComputedExpressionInteger</a> class	182
src/ <a href="#">computedExpressionString.cpp</a>	
Define the <a href="#">Tang::ComputedExpressionString</a> class	183
src/ <a href="#">error.cpp</a>	
Define the <a href="#">Tang::Error</a> class	184
src/ <a href="#">program-dumpBytecode.cpp</a>	
Define the <a href="#">Tang::Program::dumpBytecode</a> method	185
src/ <a href="#">program-execute.cpp</a>	
Define the <a href="#">Tang::Program::execute</a> method	186
src/ <a href="#">program.cpp</a>	
Define the <a href="#">Tang::Program</a> class	188
src/ <a href="#">tangBase.cpp</a>	
Define the <a href="#">Tang::TangBase</a> class	188
test/ <a href="#">test.cpp</a>	
Test the general language behaviors	189
test/ <a href="#">testGarbageCollected.cpp</a>	
Test the generic behavior of the <a href="#">Tang::GarbageCollected</a> class	190
test/ <a href="#">testSingletonObjectPool.cpp</a>	
Test the generic behavior of the <a href="#">Tang::SingletonObjectPool</a> class	191





## Chapter 5

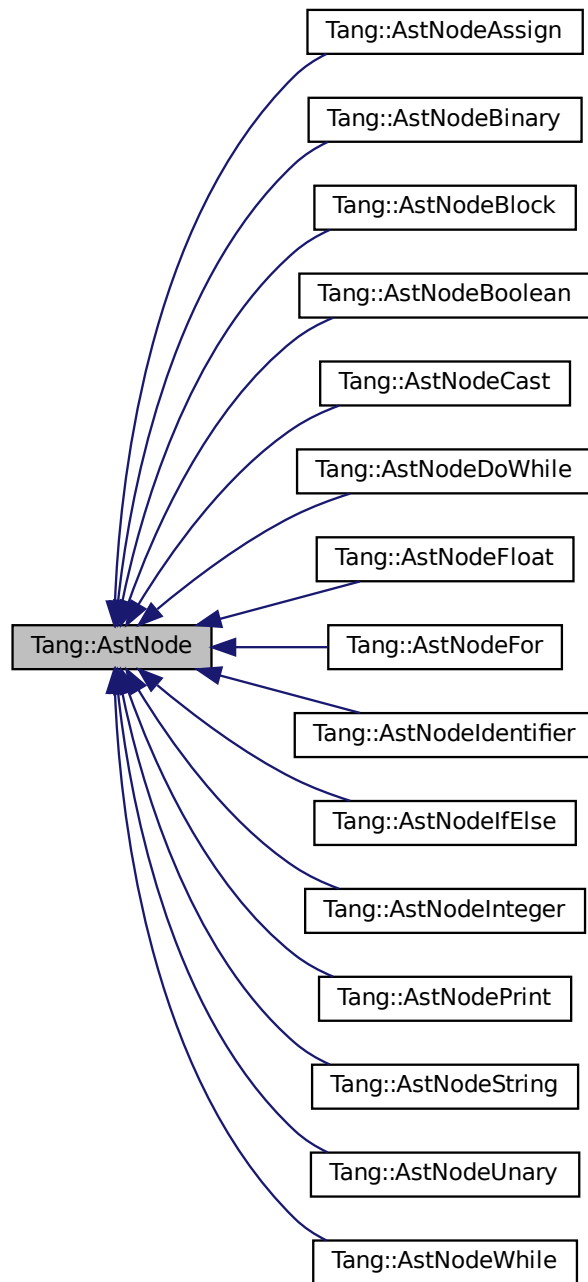
# Class Documentation

### 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 [collectIdentifiers](#) ([Program](#) &program) const  
*Compile a list of all variables in the scope.*

### 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

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

### 5.1.3 Member Function Documentation

#### 5.1.3.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

##### Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodePrint](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), and [Tang::AstNodeAssign](#).

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

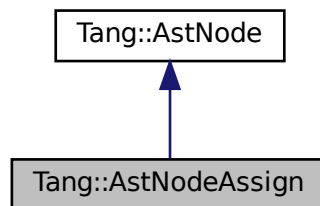
- [include/astNode.hpp](#)
- [src/astNode.cpp](#)

## 5.2 Tang::AstNodeAssign Class Reference

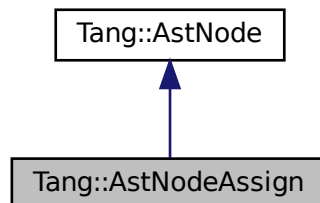
An [AstNode](#) that represents a binary expression.

```
#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 [collectIdentifiers](#) ([Program](#) &program) const override  
*Compile a list of all variables in the scope.*

### 5.2.1 Detailed Description

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

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 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.

### 5.2.3 Member Function Documentation

#### 5.2.3.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

## Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

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

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

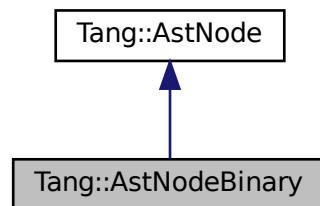
- [include/astNodeAssign.hpp](#)
- [src/astNodeAssign.cpp](#)

## 5.3 Tang::AstNodeBinary Class Reference

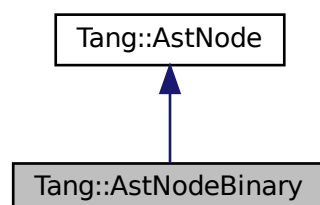
An [AstNode](#) that represents a binary expression.

```
#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](#) }

## 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 [collectIdentifiers](#) ([Program](#) &program) const override  
*Compile a list of all variables in the scope.*

### 5.3.1 Detailed Description

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

### 5.3.2 Member Enumeration Documentation

#### 5.3.2.1 Operation

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

##### 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.

### 5.3.3 Constructor & Destructor Documentation

#### 5.3.3.1 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 <a href="#">Tang::AstNodeBinary::Operation</a> 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.

### 5.3.4 Member Function Documentation

#### 5.3.4.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

##### Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

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

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

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

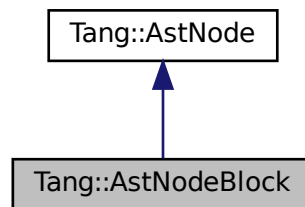
## 5.4 Tang::AstNodeBlock Class Reference

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

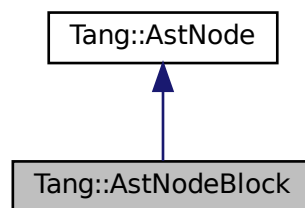


```
#include <astNodeBlock.hpp>
```

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 [collectIdentifiers](#) ([Program](#) &program) const override  
*Compile a list of all variables in the scope.*

### 5.4.1 Detailed Description

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

## 5.4.2 Constructor & Destructor Documentation

### 5.4.2.1 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.

## 5.4.3 Member Function Documentation

### 5.4.3.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

#### Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

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

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

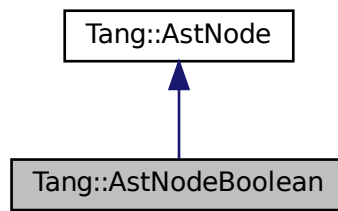
- include/[astNodeBlock.hpp](#)
- src/[astNodeBlock.cpp](#)

## 5.5 Tang::AstNodeBoolean Class Reference

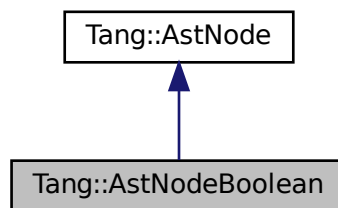
An [AstNode](#) that represents a boolean literal.

```
#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 [collectIdentifiers](#) ([Program](#) &program) const  
*Compile a list of all variables in the scope.*

### 5.5.1 Detailed Description

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

### 5.5.2 Constructor & Destructor Documentation

### 5.5.2.1 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.

## 5.5.3 Member Function Documentation

### 5.5.3.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

#### Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodePrint](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), and [Tang::AstNodeAssign](#).

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

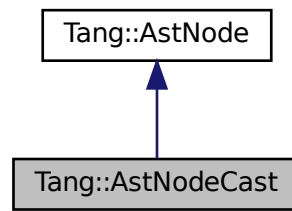
- [include/astNodeBoolean.hpp](#)
- [src/astNodeBoolean.cpp](#)

## 5.6 Tang::AstNodeCast Class Reference

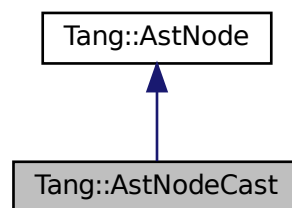
An [AstNode](#) that represents a typecast of an expression.

```
#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 [collectIdentifiers](#) ([Program](#) &program) const  
*Compile a list of all variables in the scope.*

### 5.6.1 Detailed Description

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

## 5.6.2 Member Enumeration Documentation

### 5.6.2.1 Type

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

The possible types that can be cast to.

#### Enumerator

Integer	Cast to a <a href="#">Tang::ComputedExpressionInteger</a> .
Float	Cast to a <a href="#">Tang::ComputedExpressionFloat</a> .
Boolean	Cast to a <a href="#">Tang::ComputedExpressionBoolean</a> .

## 5.6.3 Constructor & Destructor Documentation

### 5.6.3.1 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.

## 5.6.4 Member Function Documentation

### 5.6.4.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

## Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodePrint](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), and [Tang::AstNodeAssign](#).

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

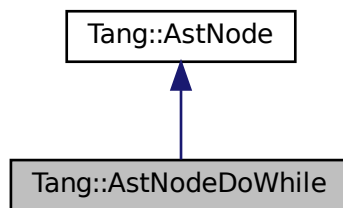
- include/[astNodeCast.hpp](#)
- src/[astNodeCast.cpp](#)

## 5.7 Tang::AstNodeDoWhile Class Reference

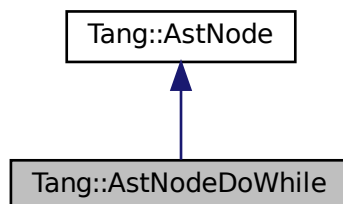
An [AstNode](#) that represents a do..while statement.

```
#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 [collectIdentifiers](#) ([Program](#) &program) const override  
*Compile a list of all variables in the scope.*

### 5.7.1 Detailed Description

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

### 5.7.2 Constructor & Destructor Documentation

#### 5.7.2.1 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.

### 5.7.3 Member Function Documentation

#### 5.7.3.1 collectIdentifiers()

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

Compile a list of all variables in the scope.



## Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

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

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

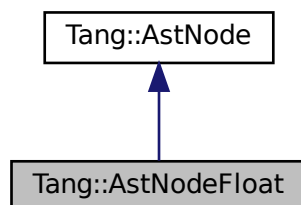
- [include/astNodeDoWhile.hpp](#)
- [src/astNodeDoWhile.cpp](#)

## 5.8 Tang::AstNodeFloat Class Reference

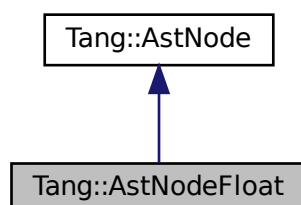
An [AstNode](#) that represents an float literal.

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

Inheritance diagram for Tang::AstNodeFloat:



Collaboration diagram for Tang::AstNodeFloat:



## Public Member Functions

- [AstNodeFloat](#) (double 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 [collectIdentifiers](#) ([Program](#) &program) const  
*Compile a list of all variables in the scope.*

### 5.8.1 Detailed Description

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

Integers are represented by the `long double` type, and so are limited in range by that of the underlying type.

### 5.8.2 Constructor & Destructor Documentation

#### 5.8.2.1 AstNodeFloat()

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

The constructor.

#### Parameters

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

### 5.8.3 Member Function Documentation

#### 5.8.3.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

## Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodePrint](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), and [Tang::AstNodeAssign](#).

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

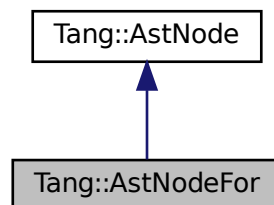
- [include/astNodeFloat.hpp](#)
- [src/astNodeFloat.cpp](#)

## 5.9 Tang::AstNodeFor Class Reference

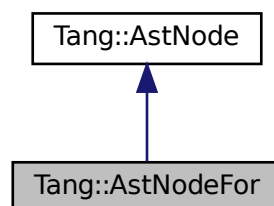
An [AstNode](#) that represents an if() statement.

```
#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 [collectIdentifiers](#) ([Program](#) &program) const override  
*Compile a list of all variables in the scope.*

### 5.9.1 Detailed Description

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

### 5.9.2 Constructor & Destructor Documentation

#### 5.9.2.1 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.

### 5.9.3 Member Function Documentation

#### 5.9.3.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

#### Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

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

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

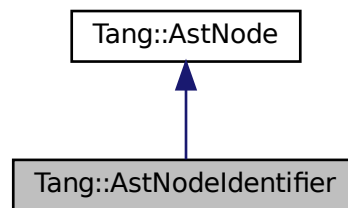
- include/[astNodeFor.hpp](#)
- src/[astNodeFor.cpp](#)

## 5.10 Tang::AstNodeIdentifier Class Reference

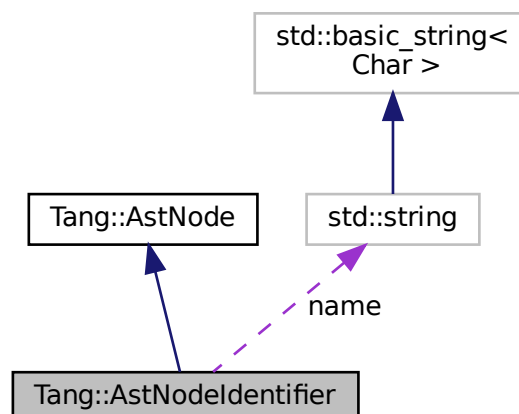
An [AstNode](#) that represents an identifier.

```
#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 [collectIdentifiers](#) ([Program](#) &program) const override  
*Compile a list of all variables in the scope.*

## Public Attributes

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

### 5.10.1 Detailed Description

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

Identifier names are represented by a string.

### 5.10.2 Constructor & Destructor Documentation

#### 5.10.2.1 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.

### 5.10.3 Member Function Documentation

## 5.10.3.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

## Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

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

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

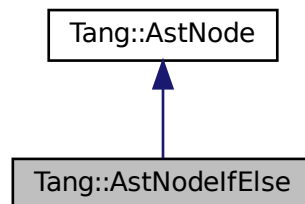
- include/[astNodeIdentifier.hpp](#)
- src/[astNodeIdentifier.cpp](#)

## 5.11 Tang::AstNodeIfElse Class Reference

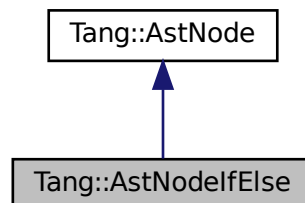
An [AstNode](#) that represents an if..else statement.

```
#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 [collectIdentifiers](#) ([Program](#) &program) const override  
*Compile a list of all variables in the scope.*

### 5.11.1 Detailed Description

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

### 5.11.2 Constructor & Destructor Documentation

#### 5.11.2.1 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.

#### 5.11.2.2 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.

### 5.11.3 Member Function Documentation

#### 5.11.3.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

## Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

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

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

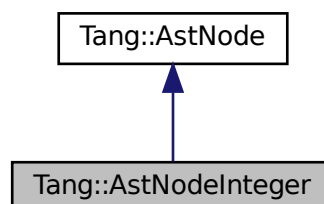
- include/[astNodeIfElse.hpp](#)
- src/[astNodeIfElse.cpp](#)

## 5.12 Tang::AstNodeInteger Class Reference

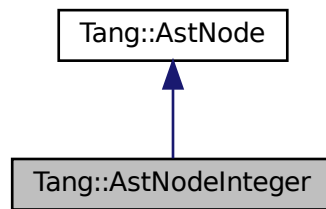
An [AstNode](#) that represents an integer literal.

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

Inheritance diagram for Tang::AstNodeInteger:



Collaboration diagram for Tang::AstNodeInteger:



## Public Member Functions

- [AstNodeInteger](#) (int64\_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 [collectIdentifiers](#) ([Program](#) &program) const  
*Compile a list of all variables in the scope.*

### 5.12.1 Detailed Description

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

Integers are represented by the `int64_t` type, and so are limited in range by that of the underlying type.

### 5.12.2 Constructor & Destructor Documentation

#### 5.12.2.1 AstNodeInteger()

```

AstNodeInteger::AstNodeInteger (
    int64_t number,
    Tang::location location )
  
```

The constructor.

#### Parameters

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

### 5.12.3 Member Function Documentation

#### 5.12.3.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

##### Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodePrint](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), and [Tang::AstNodeAssign](#).

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

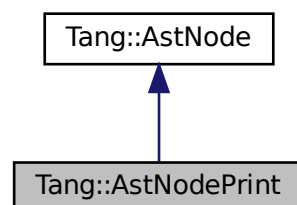
- include/[astNodeInteger.hpp](#)
- src/[astNodeInteger.cpp](#)

## 5.13 Tang::AstNodePrint Class Reference

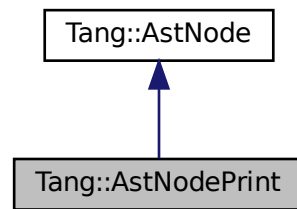
An [AstNode](#) that represents a print typeoperation.

```
#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 [collectIdentifiers](#) ([Program](#) &program) const override  
*Compile a list of all variables in the scope.*

### 5.13.1 Detailed Description

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

### 5.13.2 Member Enumeration Documentation

#### 5.13.2.1 Type

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

The type of print() requested.

## Enumerator

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

### 5.13.3 Constructor & Destructor Documentation

#### 5.13.3.1 AstNodePrint()

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

The constructor.

## Parameters

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

### 5.13.4 Member Function Documentation

#### 5.13.4.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

## Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

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

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

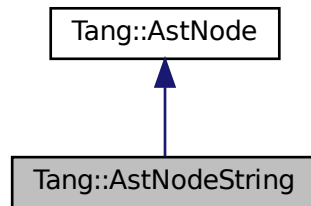
- [include/astNodePrint.hpp](#)
- [src/astNodePrint.cpp](#)

## 5.14 Tang::AstNodeString Class Reference

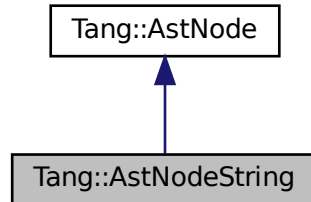
An [AstNode](#) that represents a string literal.

```
#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 [collectIdentifiers](#) ([Program](#) &program) const  
*Compile a list of all variables in the scope.*

#### 5.14.1 Detailed Description

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

## 5.14.2 Constructor & Destructor Documentation

### 5.14.2.1 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.

## 5.14.3 Member Function Documentation

### 5.14.3.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

#### Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

Reimplemented in [Tang::AstNodeWhile](#), [Tang::AstNodeUnary](#), [Tang::AstNodePrint](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), and [Tang::AstNodeAssign](#).

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

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

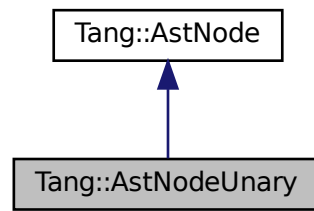
## 5.15 Tang::AstNodeUnary Class Reference

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

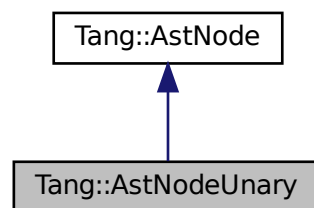
```
#include <astNodeUnary.hpp>
```



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 [collectIdentifiers](#) ([Program](#) &program) const override  
*Compile a list of all variables in the scope.*

### 5.15.1 Detailed Description

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

## 5.15.2 Member Enumeration Documentation

### 5.15.2.1 Operator

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

The type of operation.

#### Enumerator

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

## 5.15.3 Constructor & Destructor Documentation

### 5.15.3.1 AstNodeUnary()

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

The constructor.

#### Parameters

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

## 5.15.4 Member Function Documentation

### 5.15.4.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

## Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

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

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

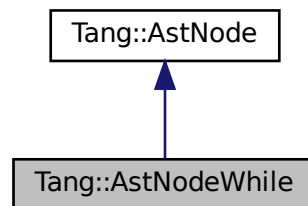
- [include/astNodeUnary.hpp](#)
- [src/astNodeUnary.cpp](#)

## 5.16 Tang::AstNodeWhile Class Reference

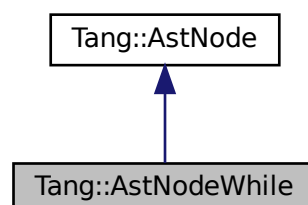
An [AstNode](#) that represents a while statement.

```
#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 [collectIdentifiers](#) ([Program](#) &program) const override  
*Compile a list of all variables in the scope.*

### 5.16.1 Detailed Description

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

### 5.16.2 Constructor & Destructor Documentation

#### 5.16.2.1 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.

### 5.16.3 Member Function Documentation

#### 5.16.3.1 collectIdentifiers()

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

Compile a list of all variables in the scope.

## Parameters

<i>program</i>	The <a href="#">Tang::Program</a> that is being compiled.
----------------	---

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

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

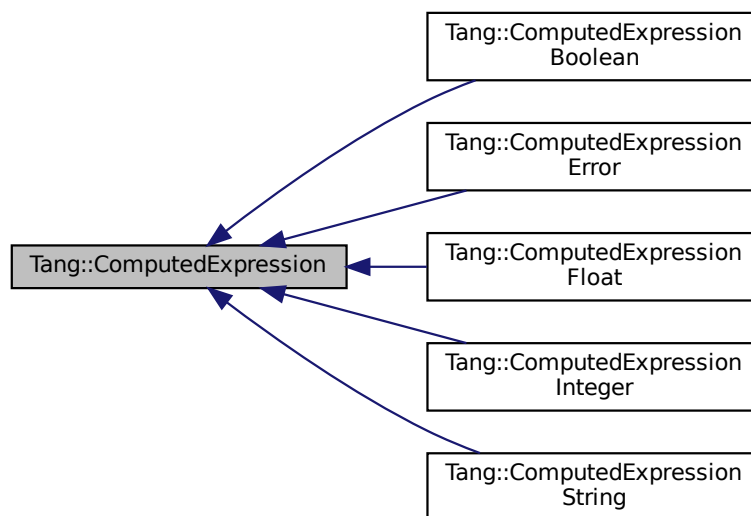
- include/[astNodeWhile.hpp](#)
- src/[astNodeWhile.cpp](#)

## 5.17 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 int &val) const

- Check whether or not the computed expression is equal to another value.*
- virtual bool [is\\_equal](#) (const double &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.*

### 5.17.1 Detailed Description

Represents the result of a computation that has been executed.

By default, it will represent a NULL value.

### 5.17.2 Member Function Documentation

#### 5.17.2.1 [\\_\\_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 <a href="#">GarbageCollected</a> value to add to this.
------------	--

## Returns

The result of the operation.

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

5.17.2.2 `__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](#).

5.17.2.3 `__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 <a href="#">GarbageCollected</a> value to divide this by.
------------	---

## Returns

The result of the operation.

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

#### 5.17.2.4 `__equal()`

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

Perform an equalit test.

##### Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to compare against.
------------	--

##### Returns

The result of the the operation.

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

#### 5.17.2.5 `__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](#).

#### 5.17.2.6 `__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](#).

#### 5.17.2.7 `__lessThan()`

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

Compute the "less than" comparison.



## Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to compare against.
------------	--

## Returns

The result of the the operation.

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

## 5.17.2.8 \_\_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 <a href="#">GarbageCollected</a> value to modulo this by.
------------	---

## Returns

The result of the operation.

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

## 5.17.2.9 \_\_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 <a href="#">GarbageCollected</a> value to multiply to this.
------------	---

## Returns

The result of the operation.

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

#### 5.17.2.10 `__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](#).

#### 5.17.2.11 `__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](#).

#### 5.17.2.12 `__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](#).

#### 5.17.2.13 `__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 <a href="#">GarbageCollected</a> value to subtract from this.
------------	---

## Returns

The result of the operation.

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

#### 5.17.2.14 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](#), and [Tang::ComputedExpressionBoolean](#).

#### 5.17.2.15 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::ComputedExpressionBoolean](#).

**5.17.2.16 is\_equal()** [2/6]

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

**5.17.2.17 is\_equal() [3/6]**

```
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](#).

**5.17.2.18 is\_equal() [4/6]**

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

#### 5.17.2.19 `is_equal()` [5/6]

```
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.

#### 5.17.2.20 `is_equal()` [6/6]

```
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](#).

#### 5.17.2.21 `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](#), and [Tang::ComputedExpressionBoolean](#).

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

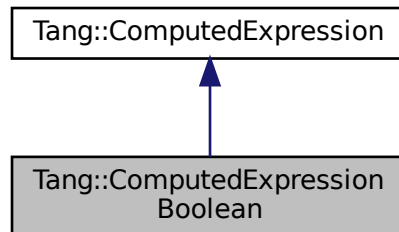
- [include/computedExpression.hpp](#)
- [src/computedExpression.cpp](#)

## 5.18 Tang::ComputedExpressionBoolean Class Reference

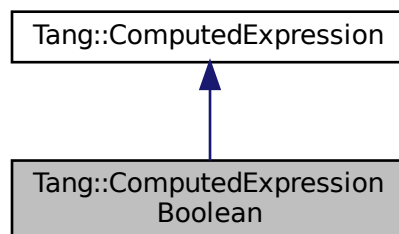
Represents an Boolean that is the result of a computation.

```
#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 equalit 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 int &val) const  
*Check whether or not the computed expression is equal to another value.*
- virtual bool `is_equal` (const double &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 __string` () const  
*Perform a type cast to string.*

### 5.18.1 Detailed Description

Represents an Boolean that is the result of a computation.

### 5.18.2 Constructor & Destructor Documentation

#### 5.18.2.1 ComputedExpressionBoolean()

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

Construct an Boolean result.



## Parameters

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

### 5.18.3 Member Function Documentation

#### 5.18.3.1 \_\_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 <a href="#">GarbageCollected</a> value to add to this.
------------	--

## Returns

The result of the operation.

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

#### 5.18.3.2 \_\_boolean()

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

Perform a type cast to boolean.

## Returns

The result of the the operation.

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

#### 5.18.3.3 \_\_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 <a href="#">GarbageCollected</a> value to divide this by.
------------	---

**Returns**

The result of the operation.

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

**5.18.3.4 `__equal()`**

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

Perform an equalit test.

**Parameters**

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to compare against.
------------	--

**Returns**

The result of the the operation.

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

**5.18.3.5 `__float()`**

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

Perform a type cast to float.

**Returns**

The result of the the operation.

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

### 5.18.3.6 \_\_integer()

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

Perform a type cast to integer.

#### Returns

The result of the the operation.

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

### 5.18.3.7 \_\_lessThan()

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

Compute the "less than" comparison.

#### Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to compare against.
------------	--

#### Returns

The result of the the operation.

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

### 5.18.3.8 \_\_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 <a href="#">GarbageCollected</a> value to modulo this by.
------------	---

#### Returns

The result of the operation.

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

### 5.18.3.9 `__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 <a href="#">GarbageCollected</a> value to multiply to this.
------------	---

#### Returns

The result of the operation.

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

### 5.18.3.10 `__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](#).

### 5.18.3.11 `__not()`

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

Compute the logical not of this value.

#### Returns

The result of the operation.

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

### 5.18.3.12 \_\_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](#).

### 5.18.3.13 \_\_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 <a href="#">GarbageCollected</a> value to subtract from this.
------------	---

#### Returns

The result of the operation.

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

### 5.18.3.14 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](#).

### 5.18.3.15 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](#).

**5.18.3.16 is\_equal() [2/6]**

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

**5.18.3.17 is\_equal() [3/6]**

```
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](#).

#### 5.18.3.18 is\_equal() [4/6]

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

#### 5.18.3.19 is\_equal() [5/6]

```
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.

#### 5.18.3.20 is\_equal() [6/6]

```
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](#).

**5.18.3.21 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:

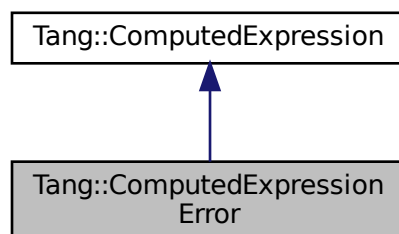
- [include/computedExpressionBoolean.hpp](#)
- [src/computedExpressionBoolean.cpp](#)

## 5.19 Tang::ComputedExpressionError Class Reference

Represents a Runtime [Error](#).

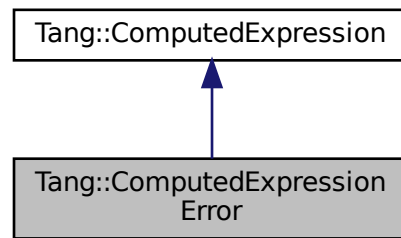
```
#include <computedExpressionError.hpp>
```

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 equalit 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 int &val)` const  
*Check whether or not the computed expression is equal to another value.*
- virtual bool `is_equal (const double &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.*

### 5.19.1 Detailed Description

Represents a Runtime [Error](#).

### 5.19.2 Constructor & Destructor Documentation

#### 5.19.2.1 ComputedExpressionError()

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

Construct a Runtime [Error](#).

##### Parameters

<code>error</code>	The <a href="#">Tang::Error</a> object.
--------------------	---

### 5.19.3 Member Function Documentation

#### 5.19.3.1 \_\_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 <a href="#">GarbageCollected</a> value to add to this.
------------	--

## Returns

The result of the operation.

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

### 5.19.3.2 \_\_boolean()

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

Perform a type cast to boolean.

## Returns

The result of the the operation.

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

### 5.19.3.3 \_\_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 <a href="#">GarbageCollected</a> value to divide this by.
------------	---

## Returns

The result of the operation.

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

### 5.19.3.4 \_\_equal()

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

Perform an equalit test.

#### Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to compare against.
------------	--

#### Returns

The result of the the operation.

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

#### 5.19.3.5 `__float()`

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

Perform a type cast to float.

#### Returns

The result of the the operation.

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

#### 5.19.3.6 `__integer()`

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

Perform a type cast to integer.

#### Returns

The result of the the operation.

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

#### 5.19.3.7 `__lessThan()`

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

Compute the "less than" comparison.

## Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to compare against.
------------	--

## Returns

The result of the the operation.

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

### 5.19.3.8 \_\_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 <a href="#">GarbageCollected</a> value to modulo this by.
------------	---

## Returns

The result of the operation.

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

### 5.19.3.9 \_\_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 <a href="#">GarbageCollected</a> value to multiply to this.
------------	---

## Returns

The result of the operation.

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

#### 5.19.3.10 `__negative()`

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

Compute the result of negating this value.

##### Returns

The result of the operation.

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

#### 5.19.3.11 `__not()`

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

Compute the logical not of this value.

##### Returns

The result of the operation.

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

#### 5.19.3.12 `__string()`

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

Perform a type cast to string.

##### Returns

The result of the the operation.

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

#### 5.19.3.13 `__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 <a href="#">GarbageCollected</a> value to subtract from this.
------------	---

## Returns

The result of the operation.

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

**5.19.3.14 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](#).

**5.19.3.15 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::ComputedExpressionBoolean](#).

**5.19.3.16 is\_equal() [2/6]**

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

**5.19.3.17 is\_equal() [3/6]**

```
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](#).

**5.19.3.18 is\_equal() [4/6]**

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



### 5.19.3.19 is\_equal() [5/6]

```
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.

### 5.19.3.20 is\_equal() [6/6]

```
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](#).

### 5.19.3.21 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:

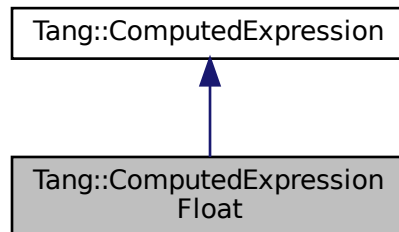
- [include/computedExpressionError.hpp](#)
- [src/computedExpressionError.cpp](#)

## 5.20 Tang::ComputedExpressionFloat Class Reference

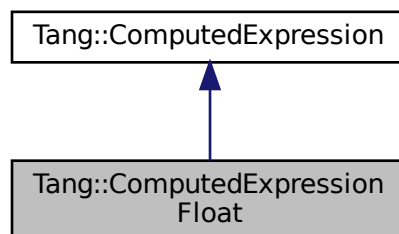
Represents a Float that is the result of a computation.

```
#include <computedExpressionFloat.hpp>
```

Inheritance diagram for Tang::ComputedExpressionFloat:



Collaboration diagram for Tang::ComputedExpressionFloat:



### Public Member Functions

- [ComputedExpressionFloat](#) (double 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 int &val) const override  
*Check whether or not the computed expression is equal to another value.*
- virtual bool [is\\_equal](#) (const double &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 equalit 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 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](#) `__modulo` (const [GarbageCollected](#) &rhs) const  
*Compute the result of moduloing this value and the supplied value.*

## Friends

- class [ComputedExpressionInteger](#)

### 5.20.1 Detailed Description

Represents a Float that is the result of a computation.

### 5.20.2 Constructor & Destructor Documentation

#### 5.20.2.1 ComputedExpressionFloat()

```
ComputedExpressionFloat::ComputedExpressionFloat (
    double val )
```

Construct a Float result.

**Parameters**

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

### 5.20.3 Member Function Documentation

#### 5.20.3.1 `__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 <a href="#">GarbageCollected</a> value to add to this.
------------	--

**Returns**

The result of the operation.

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

#### 5.20.3.2 `__boolean()`

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

Perform a type cast to boolean.

**Returns**

The result of the the operation.

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

#### 5.20.3.3 `__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 <a href="#">GarbageCollected</a> value to divide this by.
------------	---

## Returns

The result of the operation.

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

### 5.20.3.4 `__equal()`

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

Perform an equalit test.

## Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to compare against.
------------	--

## Returns

The result of the the operation.

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

### 5.20.3.5 `__float()`

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

Perform a type cast to float.

## Returns

The result of the the operation.

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

### 5.20.3.6 `__integer()`

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

Perform a type cast to integer.

#### Returns

The result of the the operation.

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

### 5.20.3.7 `__lessThan()`

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

Compute the "less than" comparison.

#### Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to compare against.
------------	--

#### Returns

The result of the the operation.

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

### 5.20.3.8 `__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 <a href="#">GarbageCollected</a> value to modulo this by.
------------	---

#### Returns

The result of the operation.

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

### 5.20.3.9 \_\_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 <a href="#">GarbageCollected</a> value to multiply to this.
------------	---

#### Returns

The result of the operation.

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

### 5.20.3.10 \_\_negative()

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

Compute the result of negating this value.

#### Returns

The result of the operation.

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

### 5.20.3.11 \_\_not()

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

Compute the logical not of this value.

#### Returns

The result of the operation.

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

### 5.20.3.12 `__string()`

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

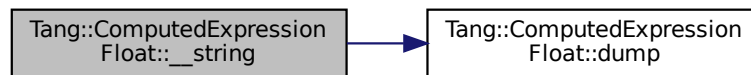
Perform a type cast to string.

#### Returns

The result of the the operation.

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

Here is the call graph for this function:



### 5.20.3.13 `__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 <a href="#">GarbageCollected</a> value to subtract from this.
------------	---

#### Returns

The result of the operation.

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

### 5.20.3.14 `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](#).

**5.20.3.15 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::ComputedExpressionBoolean](#).

**5.20.3.16 is\_equal() [2/6]**

```
bool ComputedExpressionFloat::is_equal (
    const double & 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](#).

**5.20.3.17 is\_equal() [3/6]**

```
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](#).

**5.20.3.18 is\_equal() [4/6]**

```
bool ComputedExpressionFloat::is_equal (  
    const int & 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](#).

**5.20.3.19 is\_equal() [5/6]**

```
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.

**5.20.3.20 is\_equal()** [6/6]

```
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](#).

**5.20.3.21 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:

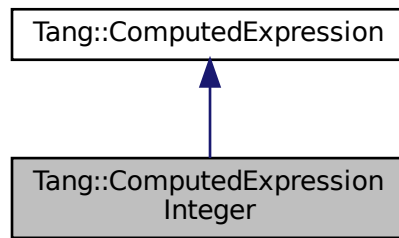
- include/[computedExpressionFloat.hpp](#)
- src/[computedExpressionFloat.cpp](#)

**5.21 Tang::ComputedExpressionInteger Class Reference**

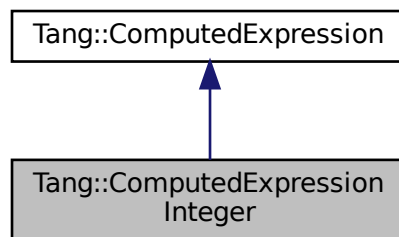
Represents an Integer that is the result of a computation.

```
#include <computedExpressionInteger.hpp>
```

Inheritance diagram for Tang::ComputedExpressionInteger:



Collaboration diagram for Tang::ComputedExpressionInteger:



## Public Member Functions

- [ComputedExpressionInteger](#) (int64\_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 int &val) const override  
*Check whether or not the computed expression is equal to another value.*
- virtual bool [is\\_equal](#) (const double &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 equalit 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 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.*

## Friends

- class [ComputedExpressionFloat](#)

### 5.21.1 Detailed Description

Represents an Integer that is the result of a computation.

### 5.21.2 Constructor & Destructor Documentation

#### 5.21.2.1 ComputedExpressionInteger()

```
ComputedExpressionInteger::ComputedExpressionInteger (
    int64_t val )
```

Construct an Integer result.

## Parameters

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

### 5.21.3 Member Function Documentation

#### 5.21.3.1 `__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 <a href="#">GarbageCollected</a> value to add to this.
------------	--

## Returns

The result of the operation.

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

#### 5.21.3.2 `__boolean()`

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

Perform a type cast to boolean.

## Returns

The result of the the operation.

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

#### 5.21.3.3 `__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 <a href="#">GarbageCollected</a> value to divide this by.
------------	---

## Returns

The result of the operation.

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

### 5.21.3.4 `__equal()`

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

Perform an equalit test.

## Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to compare against.
------------	--

## Returns

The result of the the operation.

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

### 5.21.3.5 `__float()`

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

Perform a type cast to float.

## Returns

The result of the the operation.

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

### 5.21.3.6 `__integer()`

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

Perform a type cast to integer.

#### Returns

The result of the the operation.

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

### 5.21.3.7 `__lessThan()`

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

Compute the "less than" comparison.

#### Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to compare against.
------------	--

#### Returns

The result of the the operation.

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

### 5.21.3.8 `__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 <a href="#">GarbageCollected</a> value to modulo this by.
------------	---

#### Returns

The result of the operation.

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



### 5.21.3.9 \_\_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 <a href="#">GarbageCollected</a> value to multiply to this.
------------	---

#### Returns

The result of the operation.

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

### 5.21.3.10 \_\_negative()

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

Compute the result of negating this value.

#### Returns

The result of the operation.

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

### 5.21.3.11 \_\_not()

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

Compute the logical not of this value.

#### Returns

The result of the operation.

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

**5.21.3.12 \_\_string()**

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

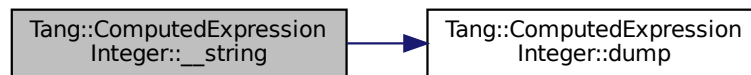
Perform a type cast to string.

**Returns**

The result of the the operation.

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

Here is the call graph for this function:

**5.21.3.13 \_\_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 <a href="#">GarbageCollected</a> value to subtract from this.
------------	---

**Returns**

The result of the operation.

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

**5.21.3.14 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](#).

**5.21.3.15 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::ComputedExpressionBoolean](#).

**5.21.3.16 is\_equal() [2/6]**

```
bool ComputedExpressionInteger::is_equal (
    const double & 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](#).

**5.21.3.17 is\_equal() [3/6]**

```
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](#).

**5.21.3.18 is\_equal() [4/6]**

```
bool ComputedExpressionInteger::is_equal (
    const int & 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](#).

**5.21.3.19 is\_equal() [5/6]**

```
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.

**5.21.3.20 is\_equal()** [6/6]

```
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](#).

**5.21.3.21 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:

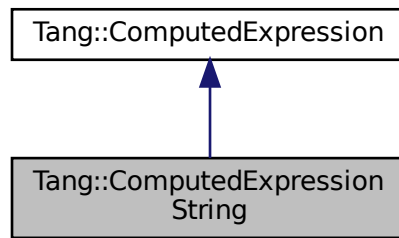
- include/[computedExpressionInteger.hpp](#)
- src/[computedExpressionInteger.cpp](#)

**5.22 Tang::ComputedExpressionString Class Reference**

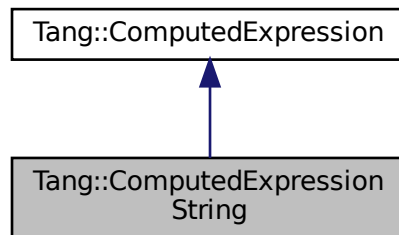
Represents a String that is the result of a computation.

```
#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 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 int &val) const  
*Check whether or not the computed expression is equal to another value.*
- virtual bool [is\\_equal](#) (const double &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 [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 \\_\\_integer](#) () const  
*Perform a type cast to integer.*
- virtual [GarbageCollected \\_\\_float](#) () const  
*Perform a type cast to float.*

### 5.22.1 Detailed Description

Represents a String that is the result of a computation.

### 5.22.2 Constructor & Destructor Documentation

#### 5.22.2.1 ComputedExpressionString()

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

Construct a String result.

#### Parameters

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

## 5.22.3 Member Function Documentation

### 5.22.3.1 `__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 <a href="#">GarbageCollected</a> value to add to this.
------------	--

#### Returns

The result of the operation.

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

### 5.22.3.2 `__boolean()`

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

Perform a type cast to boolean.

#### Returns

The result of the the operation.

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

### 5.22.3.3 `__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 <a href="#">GarbageCollected</a> value to divide this by.
------------	---



**Returns**

The result of the operation.

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

**5.22.3.4 \_\_equal()**

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

Perform an equalit test.

**Parameters**

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to compare against.
------------	--

**Returns**

The result of the the operation.

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

**5.22.3.5 \_\_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](#).

**5.22.3.6 \_\_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](#).

### 5.22.3.7 `__lessThan()`

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

Compute the "less than" comparison.

#### Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to compare against.
------------	--

#### Returns

The result of the the operation.

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

### 5.22.3.8 `__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 <a href="#">GarbageCollected</a> value to modulo this by.
------------	---

#### Returns

The result of the operation.

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

### 5.22.3.9 `__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 <a href="#">GarbageCollected</a> value to multiply to this.
------------	---

**Returns**

The result of the operation.

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

**5.22.3.10 \_\_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](#).

**5.22.3.11 \_\_not()**

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

Compute the logical not of this value.

**Returns**

The result of the operation.

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

**5.22.3.12 \_\_string()**

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

Perform a type cast to string.

**Returns**

The result of the the operation.

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

**5.22.3.13 \_\_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 <a href="#">GarbageCollected</a> value to subtract from this.
------------	---

## Returns

The result of the operation.

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

**5.22.3.14 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](#).

**5.22.3.15 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::ComputedExpressionBoolean](#).

**5.22.3.16 is\_equal() [2/6]**

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

**5.22.3.17 is\_equal() [3/6]**

```
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](#).

**5.22.3.18 is\_equal() [4/6]**

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

### 5.22.3.19 `is_equal()` [5/6]

```
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.

### 5.22.3.20 `is_equal()` [6/6]

```
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](#).

### 5.22.3.21 `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:

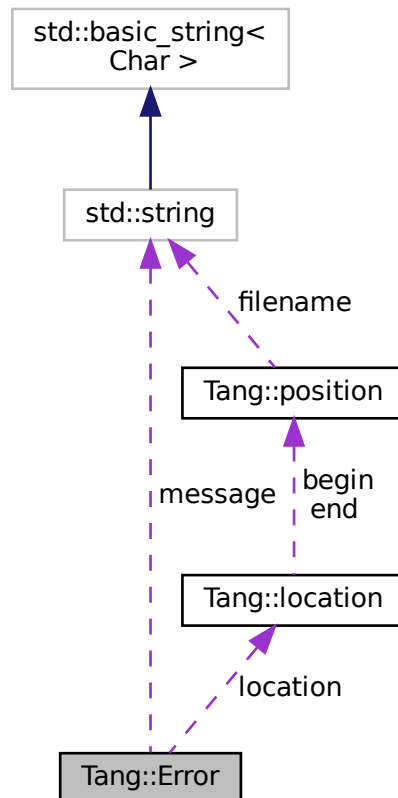
- [include/computedExpressionString.hpp](#)
- [src/computedExpressionString.cpp](#)

## 5.23 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.*

### 5.23.1 Detailed Description

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

### 5.23.2 Constructor & Destructor Documentation

#### 5.23.2.1 [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.
----------------	--------------------------------

#### 5.23.2.2 [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.

### 5.23.3 Friends And Related Function Documentation



### 5.23.3.1 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 <a href="#">Error</a> object.

#### Returns

The output stream.

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

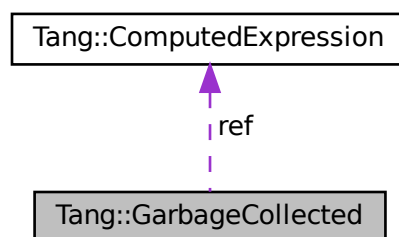
- [include/error.hpp](#)
- [src/error.cpp](#)

## 5.24 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 int &val) const  
*Compare the [GarbageCollected](#) tracked object with a supplied value.*
- bool operator== (const double &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 != 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.

### 5.24.1 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.

### 5.24.2 Constructor & Destructor Documentation

#### 5.24.2.1 *GarbageCollected*() [1/3]

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

Copy Constructor.

## Parameters

<i>The</i>	other <a href="#">GarbageCollected</a> object to copy.
------------	--

**5.24.2.2 [GarbageCollected\(\)](#) [2/3]**

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

Move Constructor.

## Parameters

<i>The</i>	other <a href="#">GarbageCollected</a> object to move.
------------	--

**5.24.2.3 [~GarbageCollected\(\)](#)**

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

Destructor.

Clean up the tracked object, if appropriate.

**5.24.2.4 [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.
-----------------	---

**5.24.3 Member Function Documentation****5.24.3.1 [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:



#### 5.24.3.2 operator"!")()

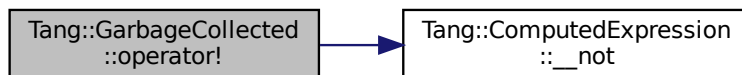
```
GarbageCollected GarbageCollected::operator! ( ) const
```

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

#### Returns

The result of the operation.

Here is the call graph for this function:



#### 5.24.3.3 operator"!=(())

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

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

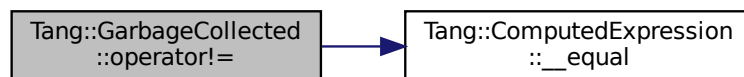
**Parameters**

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

**Returns**

The result of the operation.

Here is the call graph for this function:

**5.24.3.4 operator%()**

```

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

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

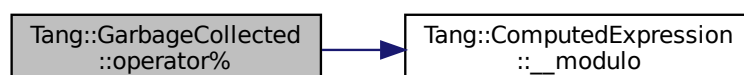
**Parameters**

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

**Returns**

The result of the operation.

Here is the call graph for this function:



**5.24.3.5 operator\*() [1/2]**

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

Access the tracked object.

**Returns**

A reference to the tracked object.

**5.24.3.6 operator\*() [2/2]**

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

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

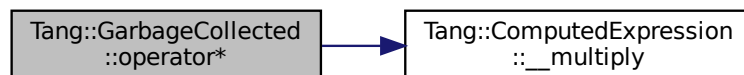
**Parameters**

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

**Returns**

The result of the operation.

Here is the call graph for this function:

**5.24.3.7 operator+()**

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

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

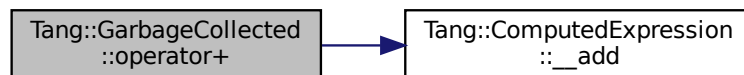
## Parameters

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

## Returns

The result of the operation.

Here is the call graph for this function:

5.24.3.8 `operator-()` [1/2]

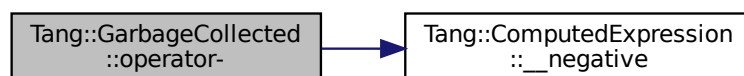
```
GarbageCollected GarbageCollected::operator- ( ) const
```

Perform a negation on the `GarbageCollected` value.

## Returns

The result of the operation.

Here is the call graph for this function:

5.24.3.9 `operator-()` [2/2]

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

Perform a subtraction between two `GarbageCollected` values.



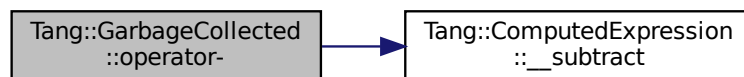
## Parameters

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

## Returns

The result of the operation.

Here is the call graph for this function:



## 5.24.3.10 operator-&gt;()

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

Access the tracked object as a pointer.

## Returns

A pointer to the tracked object.

## 5.24.3.11 operator/()

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

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

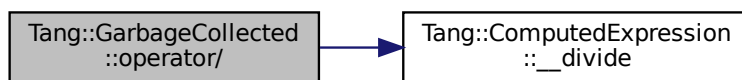
## Parameters

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

## Returns

The result of the operation.

Here is the call graph for this function:



#### 5.24.3.12 operator<()

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

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

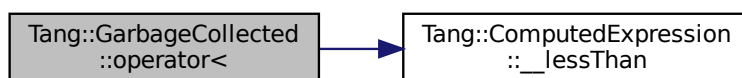
##### Parameters

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

##### Returns

The result of the operation.

Here is the call graph for this function:



#### 5.24.3.13 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.

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

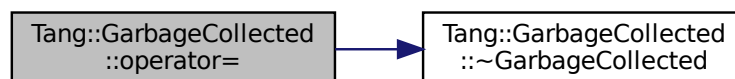
```
GarbageCollected& Tang::GarbageCollected::operator= (
    const GarbageCollected & other ) [inline]
```

Copy Assignment.

## Parameters

<i>The</i>	other <a href="#">GarbageCollected</a> object.
------------	--

Here is the call graph for this function:

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

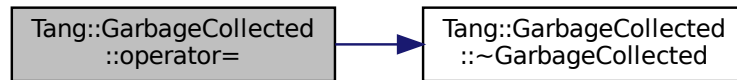
```
GarbageCollected& Tang::GarbageCollected::operator= (
    GarbageCollected && other ) [inline]
```

Move Assignment.

## Parameters

<i>The</i>	other <a href="#">GarbageCollected</a> object.
------------	--

Here is the call graph for this function:



#### 5.24.3.16 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.

#### 5.24.3.17 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.

### 5.24.3.18 operator==( ) [3/8]

```
bool GarbageCollected::operator== (
    const double & 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.

### 5.24.3.19 operator==( ) [4/8]

```
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.

### 5.24.3.20 operator==( ) [5/8]

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

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

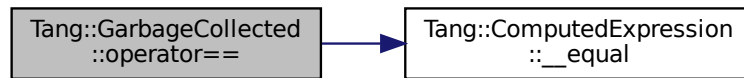
#### Parameters

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

#### Returns

The result of the operation.

Here is the call graph for this function:



#### 5.24.3.21 operator==( ) [6/8]

```
bool GarbageCollected::operator== (
    const int & 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.

#### 5.24.3.22 operator==( ) [7/8]

```
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.

### 5.24.3.23 operator==( ) [8/8]

```
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.

### 5.24.3.24 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.

### 5.24.3.25 operator>=( )

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

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

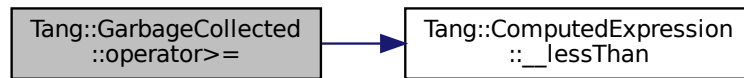
#### Parameters

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

#### Returns

The result of the operation.

Here is the call graph for this function:



## 5.24.4 Friends And Related Function Documentation

### 5.24.4.1 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 <a href="#">GarbageCollected</a> value.

#### Returns

The output stream.

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

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

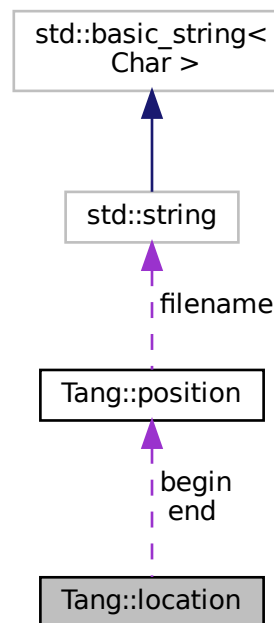
## 5.25 Tang::location Class Reference

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.*

### 5.25.1 Detailed Description

Two points in a source file.

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

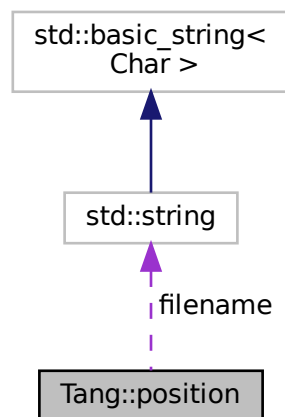
- [build/generated/location.hh](#)

## 5.26 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.*

### 5.26.1 Detailed Description

A point in a source file.

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

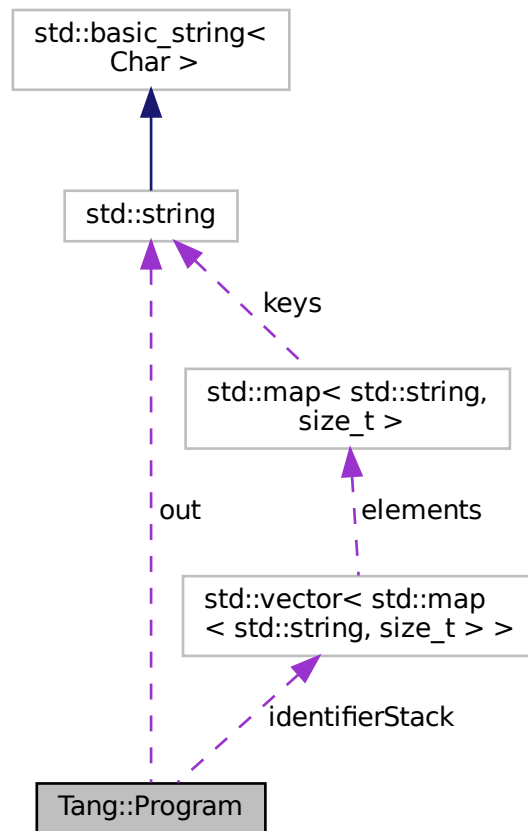
- `build/generated/location.hh`

## 5.27 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 (uint64_t)`  
*Add a uint64\_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, uint64_t jumpTarget)`  
*Set the target address of a Jump opcode.*

## Public Attributes

- `std::string out`  
*The output of the program, resulting from the program execution.*
- `std::vector< std::map< std::string, size_t > > identifierStack`  
*Stack of mappings of identifiers to their stack locations.*

### 5.27.1 Detailed Description

Represents a compiled script or template that may be executed.

### 5.27.2 Member Enumeration Documentation

#### 5.27.2.1 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.

### 5.27.3 Constructor & Destructor Documentation

#### 5.27.3.1 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> .

## 5.27.4 Member Function Documentation

### 5.27.4.1 `addBytecode()`

```
size_t Program::addBytecode (
    uint64_t op )
```

Add a `uint64_t` to the Bytecode.

## Parameters

<i>op</i>	The value to add to the Bytecode.
-----------	-----------------------------------

## Returns

The size of the bytecode structure.

### 5.27.4.2 `dumpBytecode()`

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

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

## Returns

A string containing the Opcode representation.

### 5.27.4.3 `execute()`

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

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

## Returns

The current [Program](#) object.

#### 5.27.4.4 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.

#### 5.27.4.5 getBytecode()

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

Get the Bytecode vector.

##### Returns

The Bytecode vector.

#### 5.27.4.6 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.

#### 5.27.4.7 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.

#### 5.27.4.8 setJumpTarget()

```
bool Program::setJumpTarget (
    size_t opcodeAddress,
    uint64_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.

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](#)

## 5.28 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.*

#### 5.28.1 Detailed Description

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

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

#### 5.28.2 Member Function Documentation



### 5.28.2.1 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.

### 5.28.2.2 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.

### 5.28.2.3 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](#)

## 5.29 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](#).*

### 5.29.1 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).

### 5.29.2 Constructor & Destructor Documentation

#### 5.29.2.1 TangBase()

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

The constructor.

Isn't it glorious.

### 5.29.3 Member Function Documentation

#### 5.29.3.1 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:

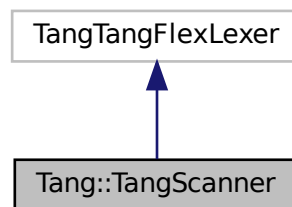
- include/[tangBase.hpp](#)
- src/[tangBase.cpp](#)

## 5.30 Tang::TangScanner Class Reference

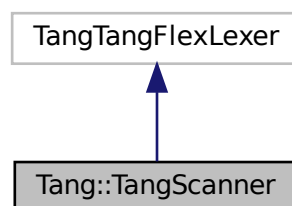
The Flex lexer class for the main Tang language.

```
#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.*

### 5.30.1 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.

### 5.30.2 Constructor & Destructor Documentation

#### 5.30.2.1 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)

### 5.30.3 Member Function Documentation

#### 5.30.3.1 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

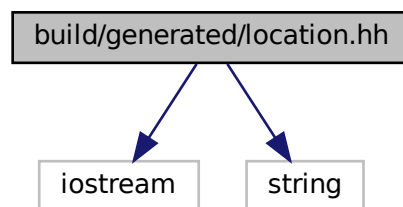
### 6.1 build/generated/location.hh File Reference

Define the Tang ::location class.

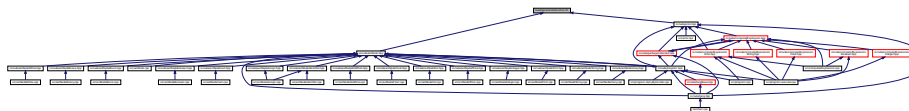
```
#include <iostream>
```

```
#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&lt;&lt;() [2/2]

```
template<typename YYChar >
std::basic_ostream<YYChar>& Tang::operator<< (
    std::basic_ostream< YYChar > & ostr,
    const position & pos )
```

Intercept output stream redirection.

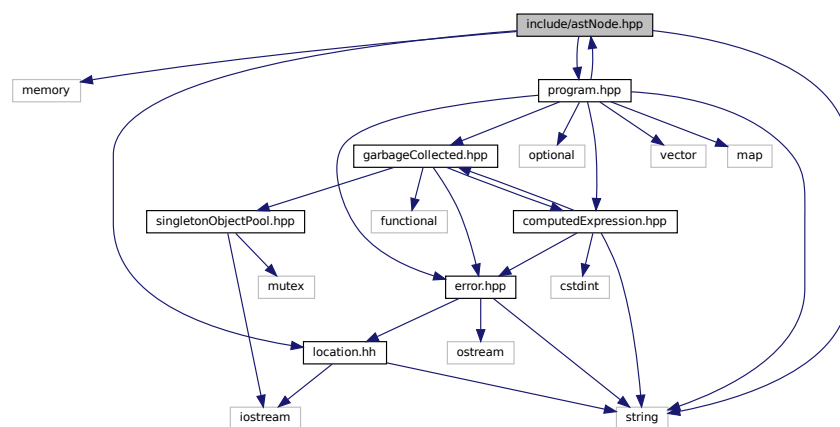
## Parameters

<i>ostr</i>	the destination output stream
<i>pos</i>	a reference to the position to redirect

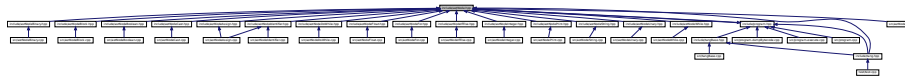
## 6.2 include/astNode.hpp File Reference

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

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

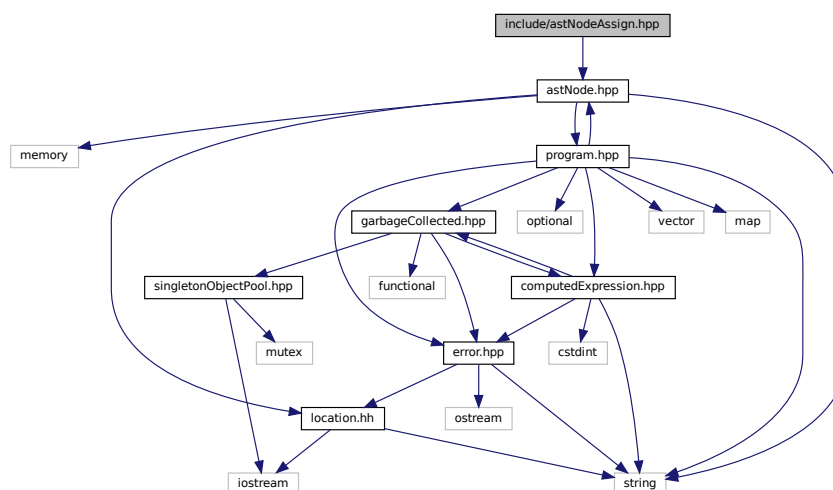
Declare the [Tang::AstNode](#) base class.

## 6.3 include/astNodeAssign.hpp File Reference

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

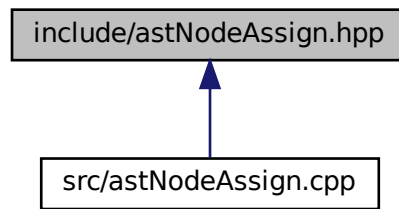
```
#include "astNode.hpp"
```

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.*

### 6.3.1 Detailed Description

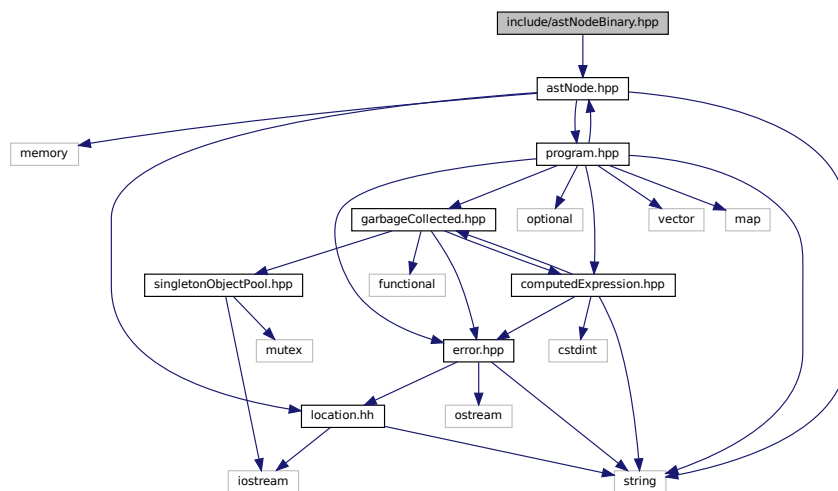
Declare the [Tang::AstNodeAssign](#) class.

## 6.4 include/astNodeBinary.hpp File Reference

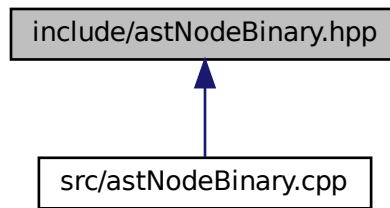
Declare the [Tang::AstNodeBinary](#) class.

```
#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.

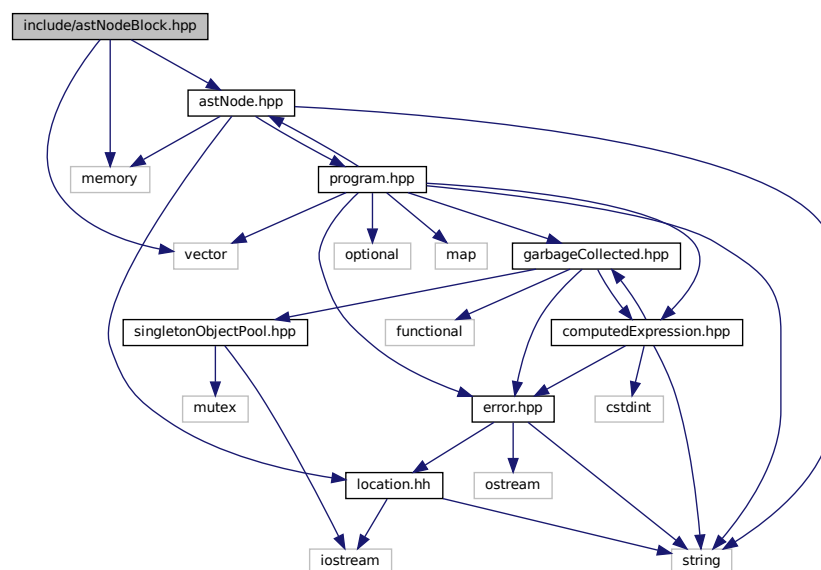
### 6.4.1 Detailed Description

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

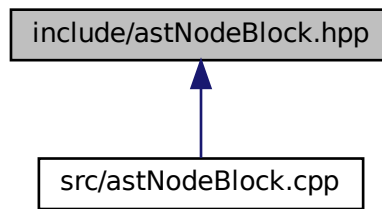
## 6.5 include/astNodeBlock.hpp File Reference

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.

### 6.5.1 Detailed Description

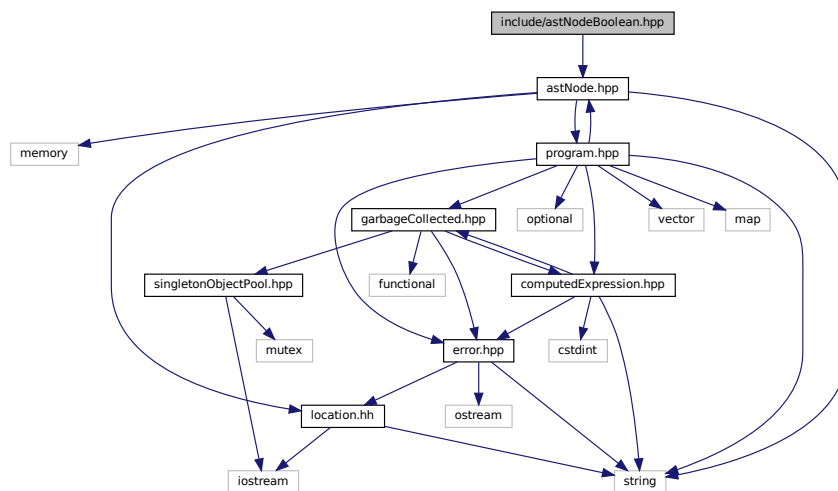
Declare the [Tang::AstNodeBlock](#) class.

## 6.6 include/astNodeBoolean.hpp File Reference

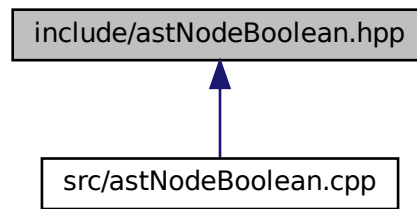
Declare the [Tang::AstNodeBoolean](#) class.

```
#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.

### 6.6.1 Detailed Description

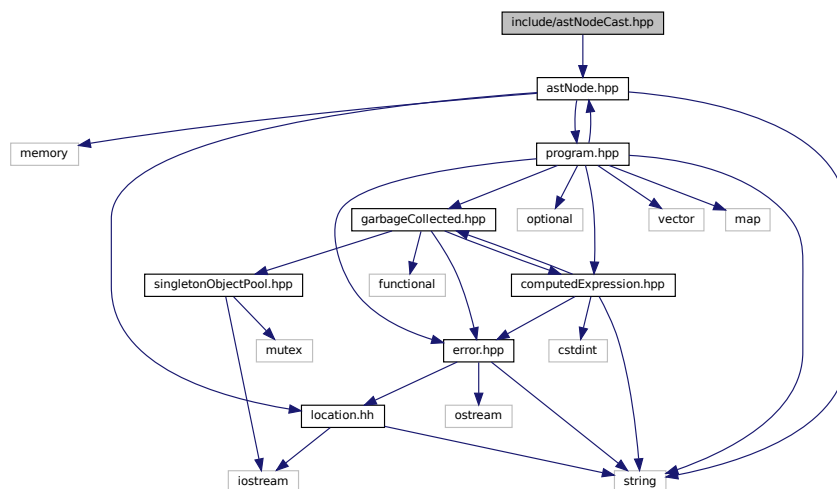
Declare the [Tang::AstNodeBoolean](#) class.

## 6.7 include/astNodeCast.hpp File Reference

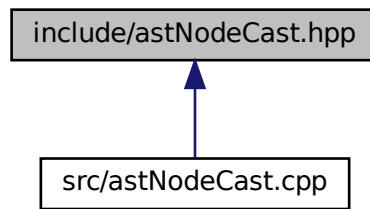
Declare the [Tang::AstNodeCast](#) class.

```
#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.

### 6.7.1 Detailed Description

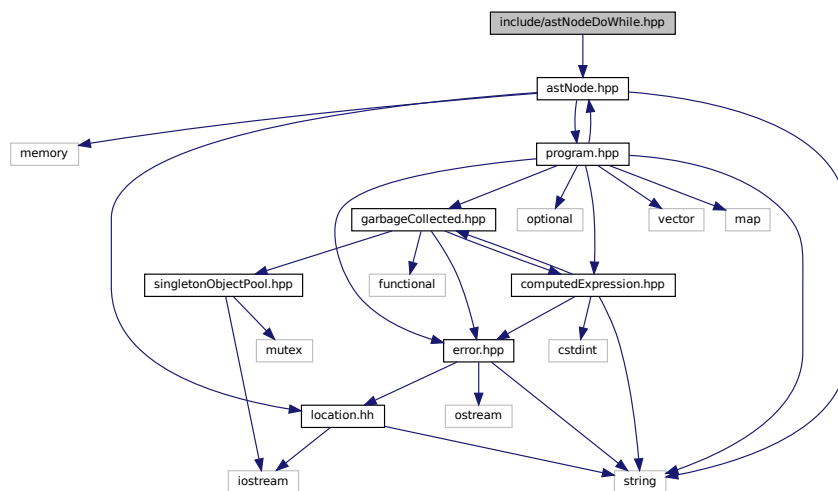
Declare the [Tang::AstNodeCast](#) class.

## 6.8 include/astNodeDoWhile.hpp File Reference

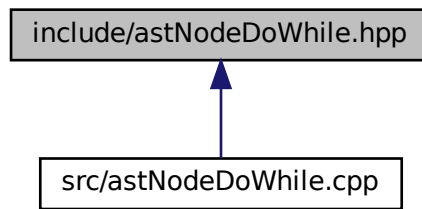
Declare the [Tang::AstNodeDoWhile](#) class.

```
#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.

### 6.8.1 Detailed Description

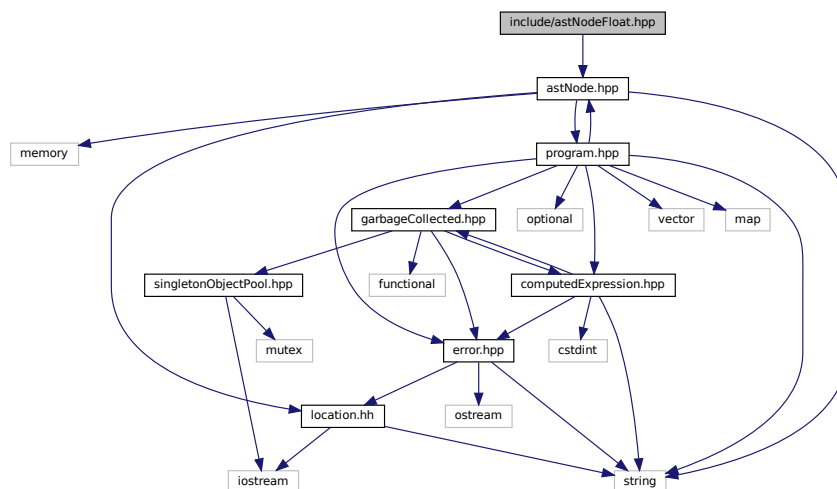
Declare the [Tang::AstNodeDoWhile](#) class.

## 6.9 include/astNodeFloat.hpp File Reference

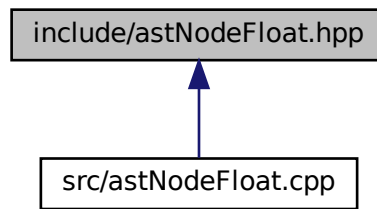
Declare the [Tang::AstNodeFloat](#) class.

```
#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.

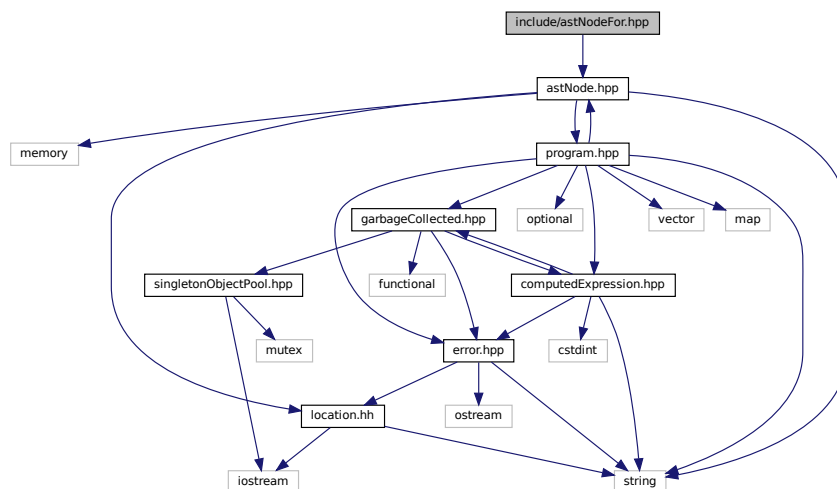
### 6.9.1 Detailed Description

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

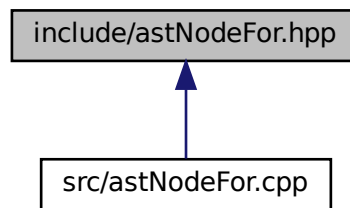
## 6.10 include/astNodeFor.hpp File Reference

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.

### 6.10.1 Detailed Description

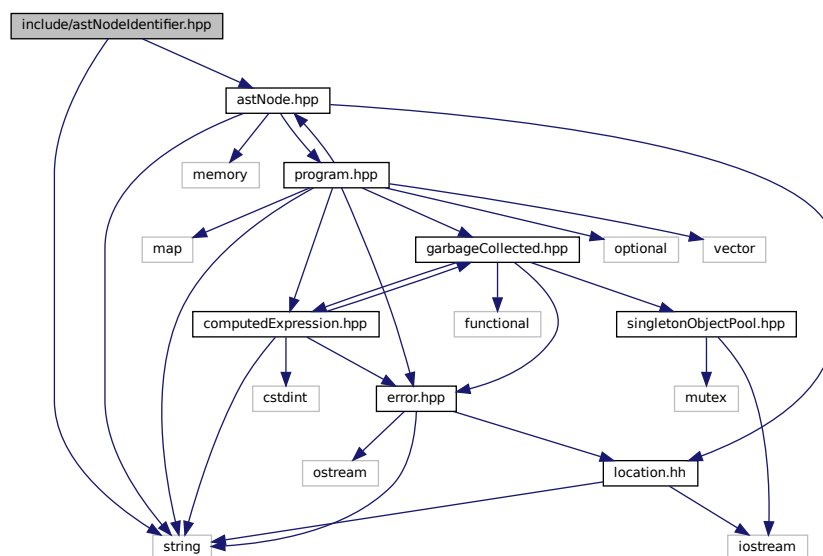
Declare the [Tang::AstNodeFor](#) class.

## 6.11 include/astNodeIdentifier.hpp File Reference

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

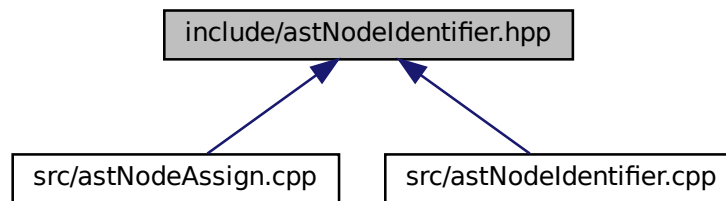
```
#include <string>
#include "astNode.hpp"
```

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.

### 6.11.1 Detailed Description

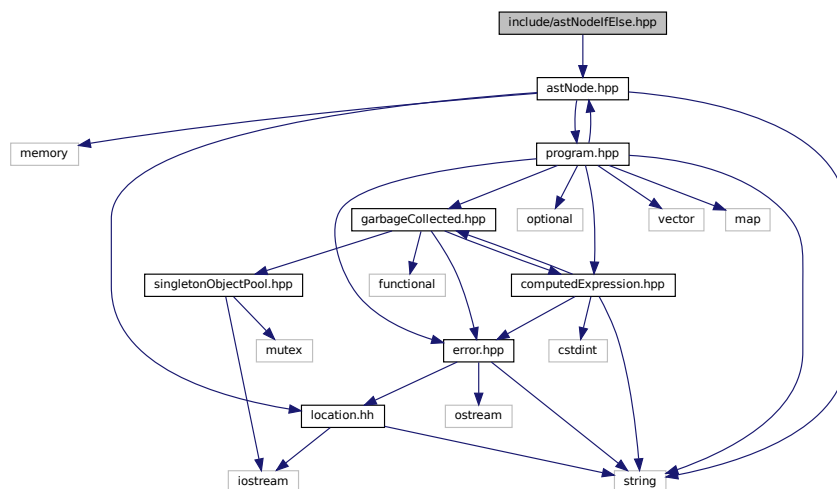
Declare the [Tang::AstNodeIdentifier](#) class.

## 6.12 include/astNodeIfElse.hpp File Reference

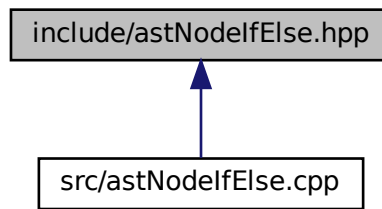
Declare the [Tang::AstNodeIfElse](#) class.

```
#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.

### 6.12.1 Detailed Description

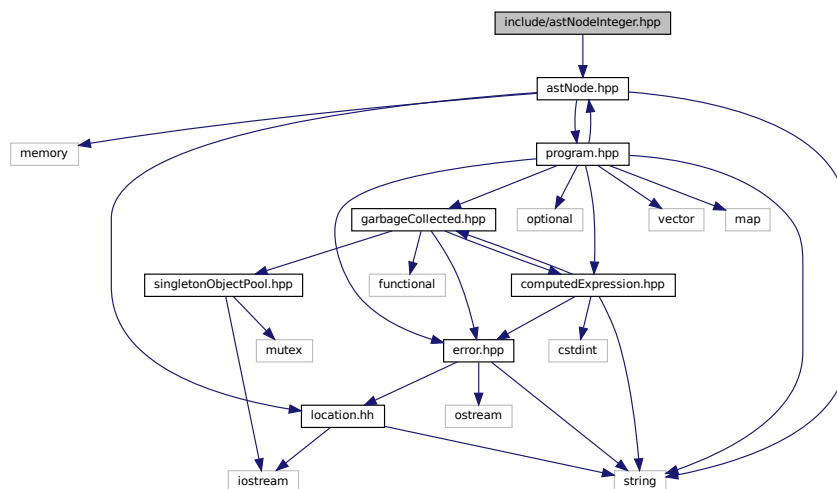
Declare the [Tang::AstNodeIfElse](#) class.

## 6.13 include/astNodeInteger.hpp File Reference

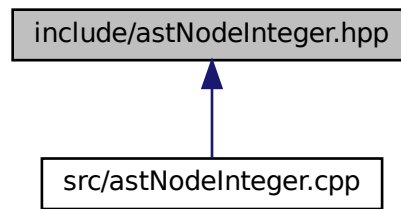
Declare the [Tang::AstNodeInteger](#) class.

```
#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.

### 6.13.1 Detailed Description

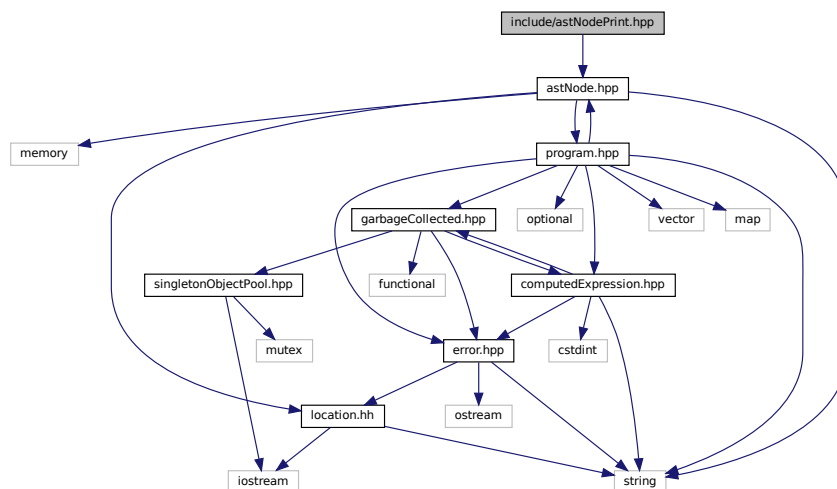
Declare the [Tang::AstNodeInteger](#) class.

## 6.14 include/astNodePrint.hpp File Reference

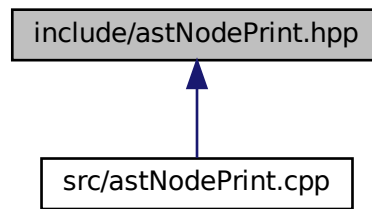
Declare the [Tang::AstNodePrint](#) class.

```
#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.

### 6.14.1 Detailed Description

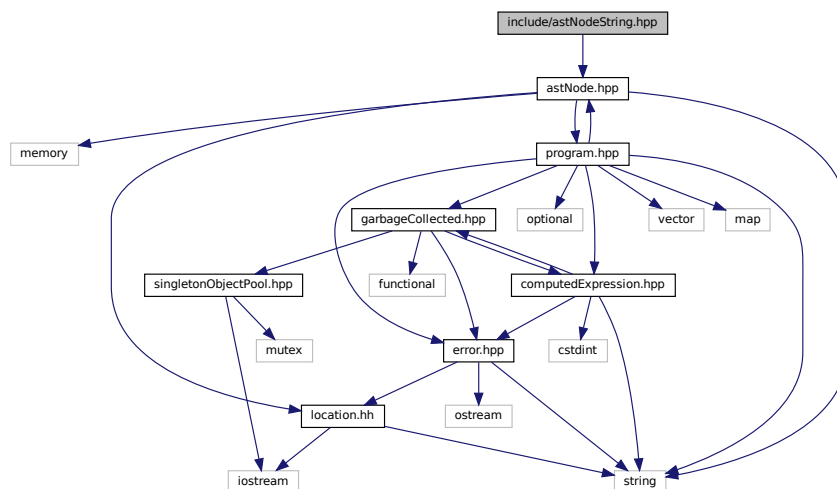
Declare the [Tang::AstNodePrint](#) class.

## 6.15 include/astNodeString.hpp File Reference

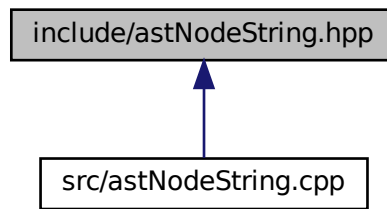
Declare the [Tang::AstNodeString](#) class.

```
#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.

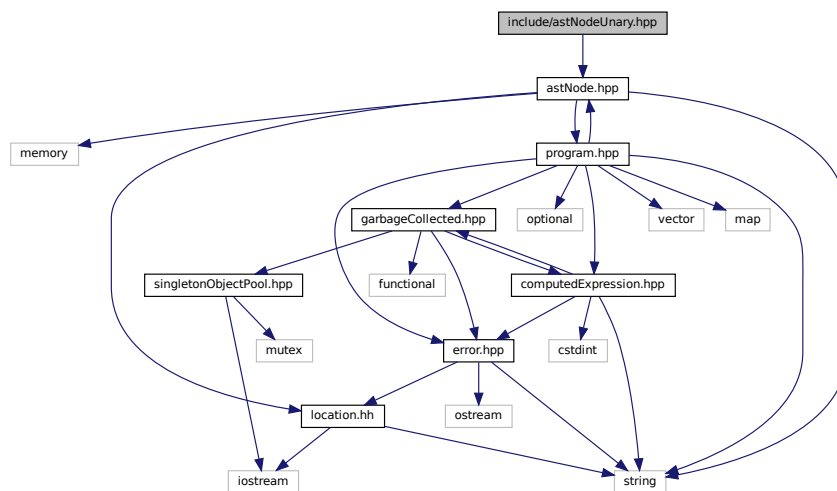
### 6.15.1 Detailed Description

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

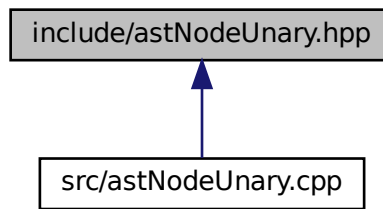
## 6.16 include/astNodeUnary.hpp File Reference

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.

### 6.16.1 Detailed Description

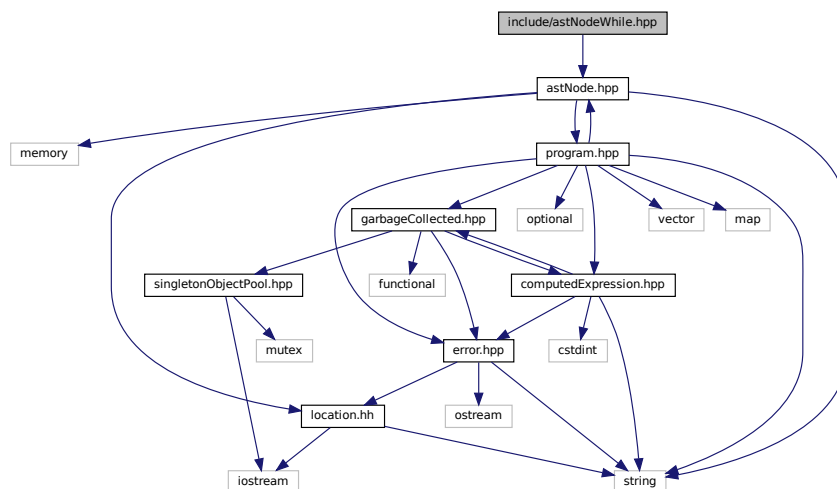
Declare the [Tang::AstNodeUnary](#) class.

## 6.17 include/astNodeWhile.hpp File Reference

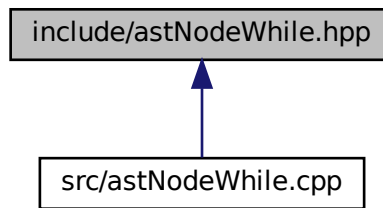
Declare the [Tang::AstNodeWhile](#) class.

```
#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.

### 6.17.1 Detailed Description

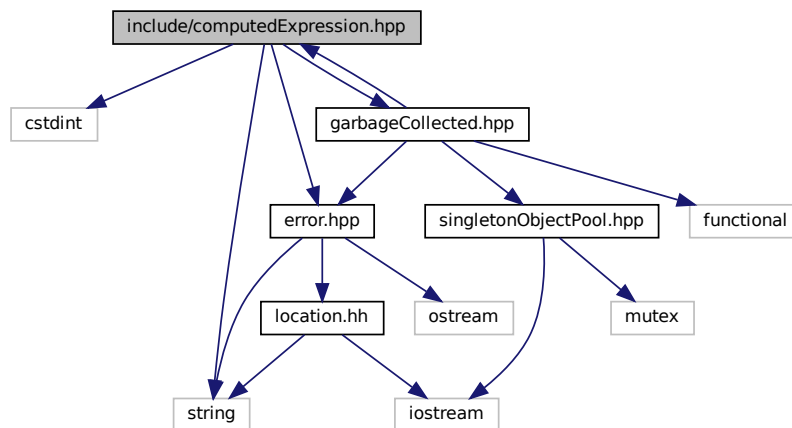
Declare the [Tang::AstNodeWhile](#) class.

## 6.18 include/computedExpression.hpp File Reference

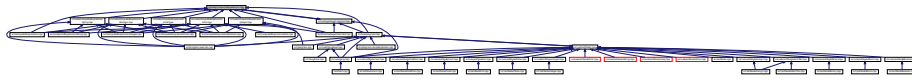
Declare the [Tang::ComputedExpression](#) base class.

```
#include <stdint>
#include <string>
#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.*

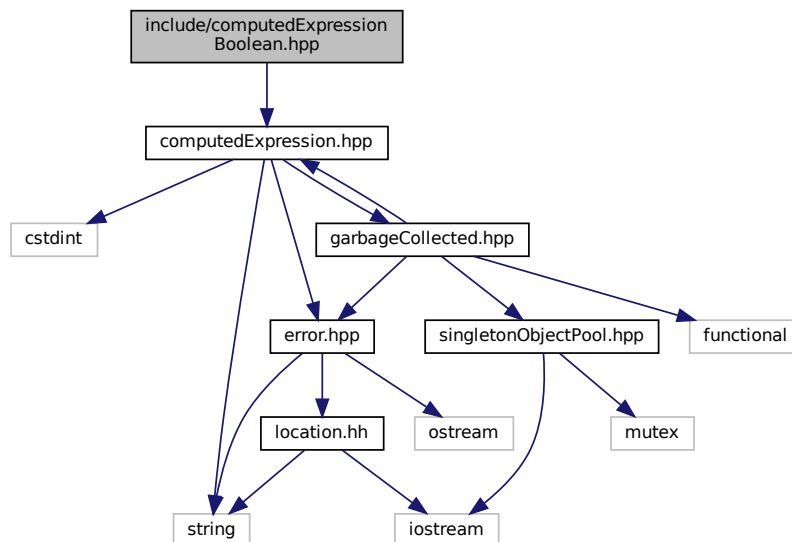
### 6.18.1 Detailed Description

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

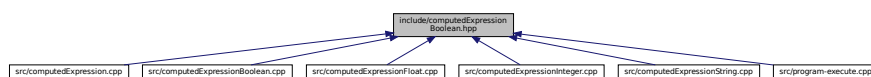
## 6.19 include/computedExpressionBoolean.hpp File Reference

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.*

### 6.19.1 Detailed Description

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

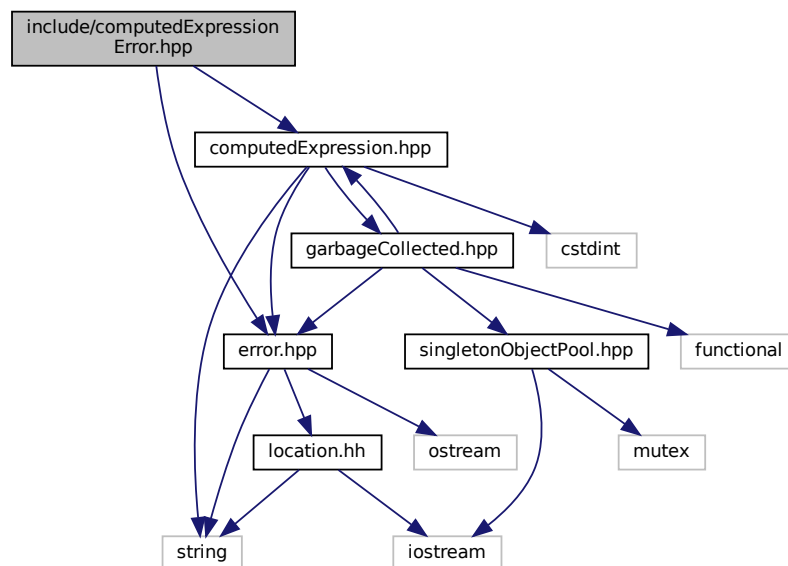
## 6.20 include/computedExpressionError.hpp File Reference

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

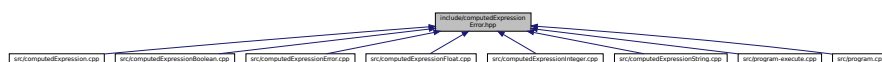
```
#include "computedExpression.hpp"
```

```
#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.*

### 6.20.1 Detailed Description

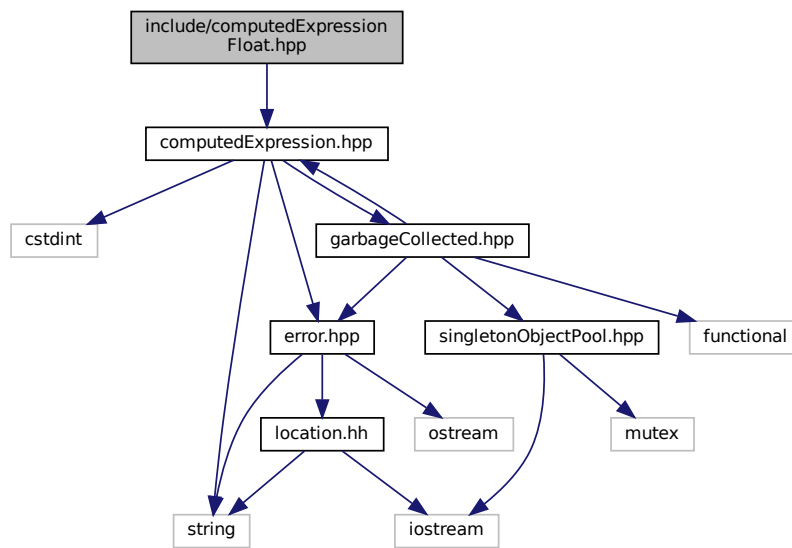
Declare the [Tang::ComputedExpressionError](#) class.

## 6.21 include/computedExpressionFloat.hpp File Reference

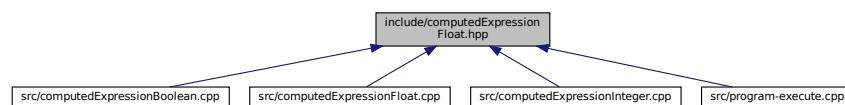
Declare the [Tang::ComputedExpressionFloat](#) class.

```
#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.*

### 6.21.1 Detailed Description

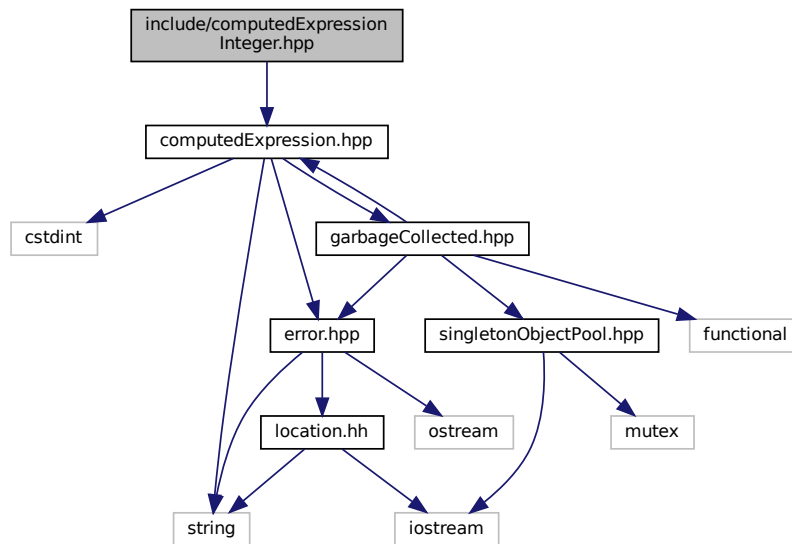
Declare the [Tang::ComputedExpressionFloat](#) class.

## 6.22 include/computedExpressionInteger.hpp File Reference

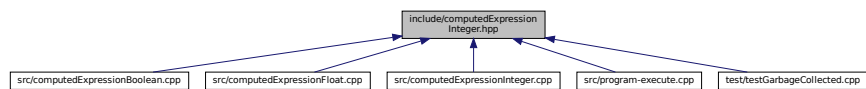
Declare the [Tang::ComputedExpressionInteger](#) class.

```
#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.*

#### 6.22.1 Detailed Description

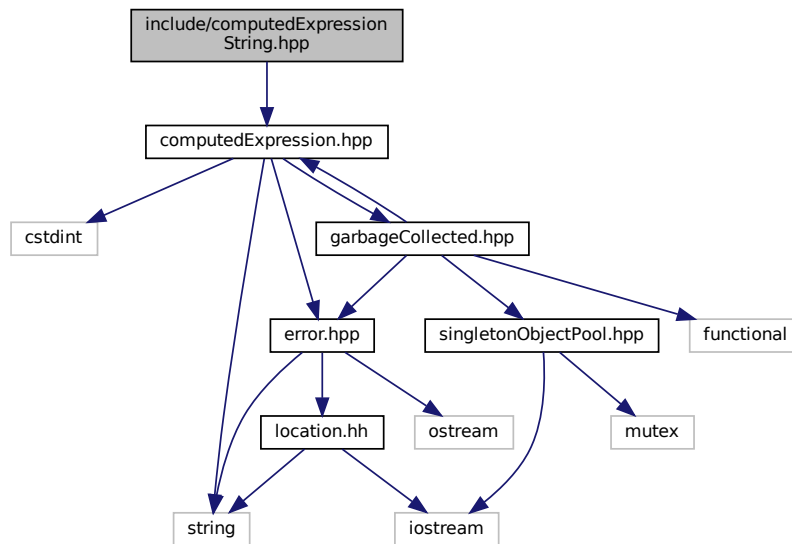
Declare the [Tang::ComputedExpressionInteger](#) class.

## 6.23 include/computedExpressionString.hpp File Reference

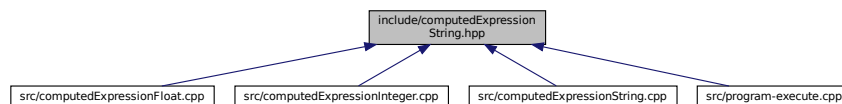
Declare the [Tang::ComputedExpressionString](#) class.

```
#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.*

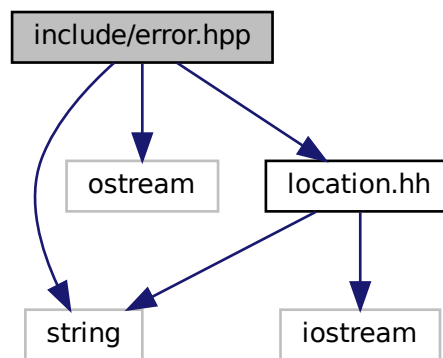
### 6.23.1 Detailed Description

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

## 6.24 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.*

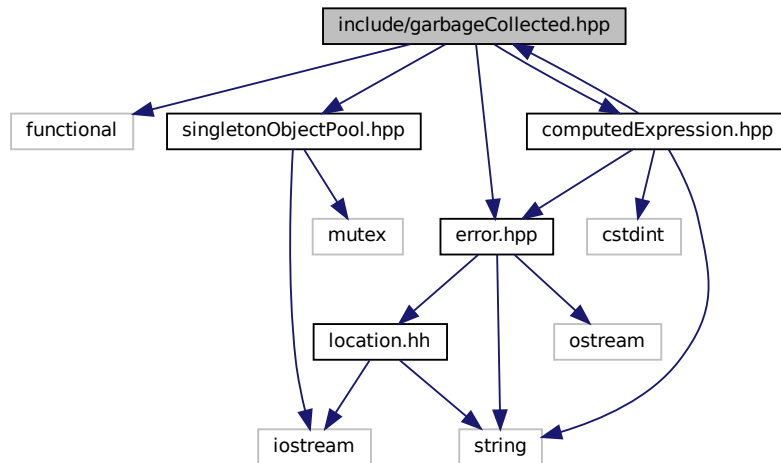
#### 6.24.1 Detailed Description

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

## 6.25 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.*

### 6.25.1 Detailed Description

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

## 6.26 include/macros.hpp File Reference

Contains generic macros.

## Macros

- `#define TANG_UNUSED(x) x`

*Instruct the compiler that a function argument will not be used so that it does not generate an error.*

### 6.26.1 Detailed Description

Contains generic macros.

### 6.26.2 Macro Definition Documentation

#### 6.26.2.1 TANG\_UNUSED

```
#define TANG_UNUSED(  
    x ) x
```

Instruct the compiler that a function argument will not be used so that it does not generate an error.

When defining a function, use the `TANG_UNUSED()` macro around any argument which is *not* used in the function, in order to squash any compiler warnings. e.g., `void foo(int TANG_UNUSED(a)) {}`

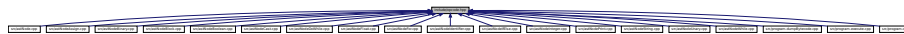
#### Parameters

x	The argument to be ignored.
---	-----------------------------

## 6.27 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_POP` , `JMPT_POP` , `NULLVAL` , `INTEGER` ,  
`FLOAT` , `BOOLEAN` , `STRING` , `ADD` ,  
`SUBTRACT` , `MULTIPLY` , `DIVIDE` , `MODULO` ,  
`NEGATIVE` , `NOT` , `LT` , `LTE` ,  
`GT` , `GTE` , `EQ` , `NEQ` ,  
`CASTINTEGER` , `CASTFLOAT` , `CASTBOOLEAN` , `PRINT` }

### 6.27.1 Detailed Description

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

### 6.27.2 Enumeration Type Documentation

#### 6.27.2.1 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_POP	PC #: pop val, if false, 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.
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.
CASTINTEGER	Pop a val, typecast to int, push.
CASTFLOAT	Pop a val, typecast to float, push.
CASTBOOLEAN	Pop a val, typecast to boolean, push.
PRINT	Pop val, print(val), push error or NULL.



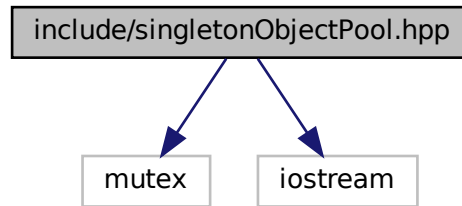


## 6.29 include/singletonObjectPool.hpp File Reference

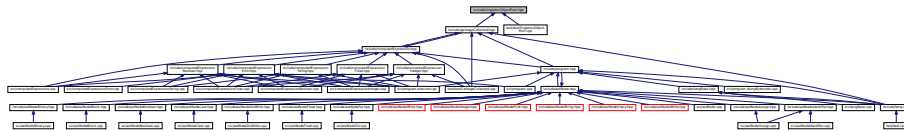
Declare the [Tang::SingletonObjectPool](#) class.

```
#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.*

#### 6.29.1 Detailed Description

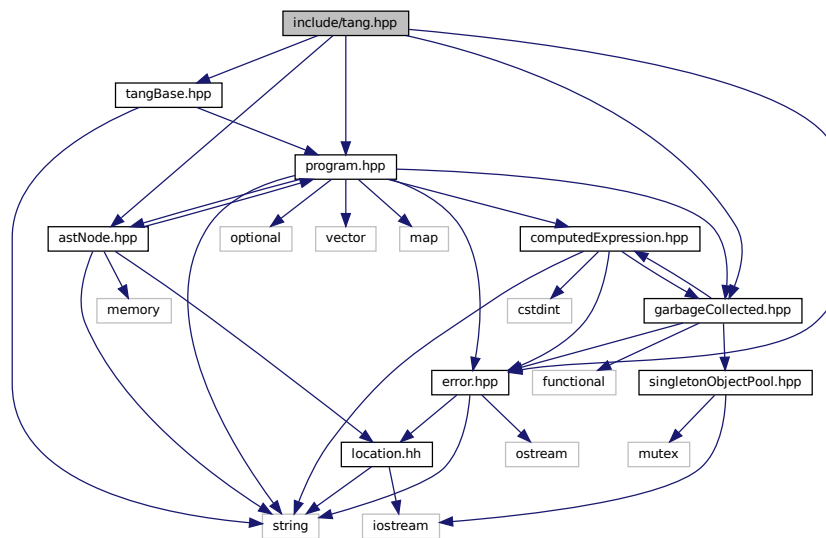
Declare the [Tang::SingletonObjectPool](#) class.

## 6.30 include/tang.hpp File Reference

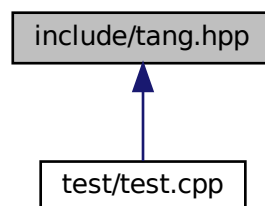
Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

```
#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:



### 6.30.1 Detailed Description

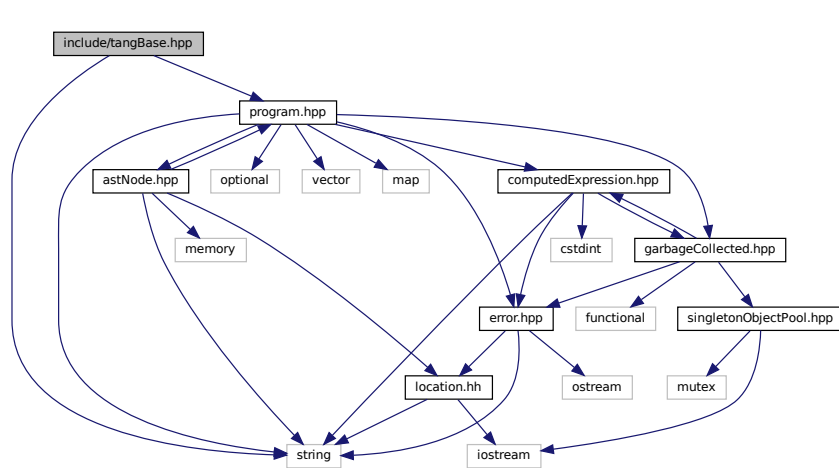
Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

## 6.31 include/tangBase.hpp File Reference

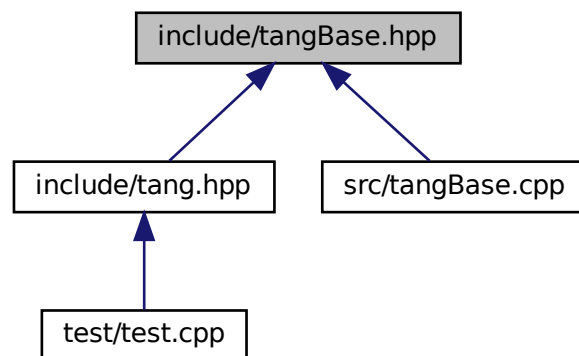
Declare the [Tang::TangBase](#) class used to interact with Tang.

```
#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.*

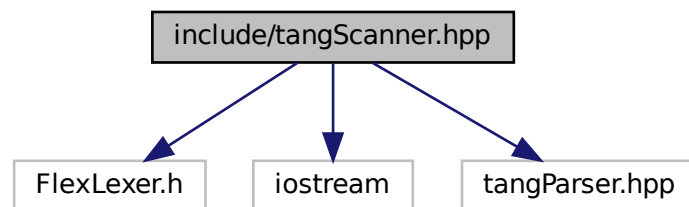
### 6.31.1 Detailed Description

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

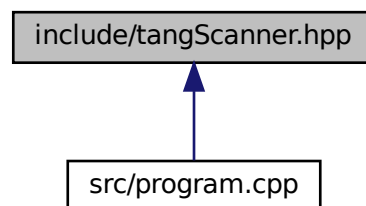
## 6.32 include/tangScanner.hpp File Reference

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()`

### 6.32.1 Detailed Description

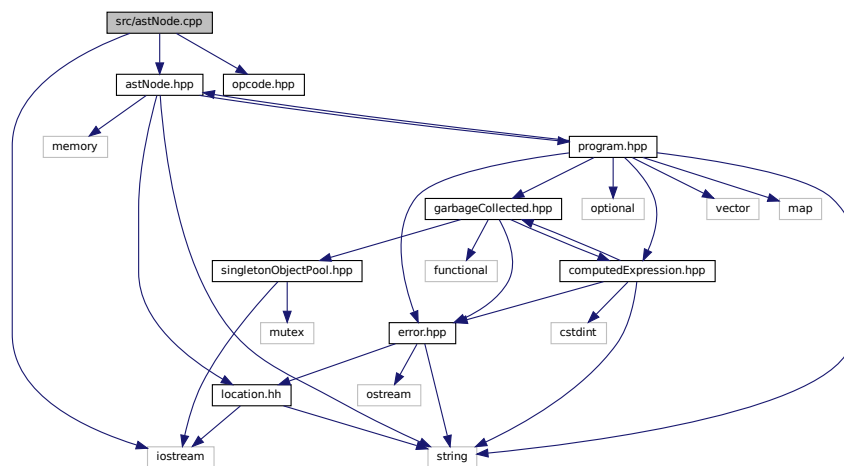
Declare the [Tang::TangScanner](#) used to tokenize a Tang script.

## 6.33 src/astNode.cpp File Reference

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

```
#include <iostream>
#include "astNode.hpp"
#include "opcode.hpp"
```

Include dependency graph for astNode.cpp:



### 6.33.1 Detailed Description

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

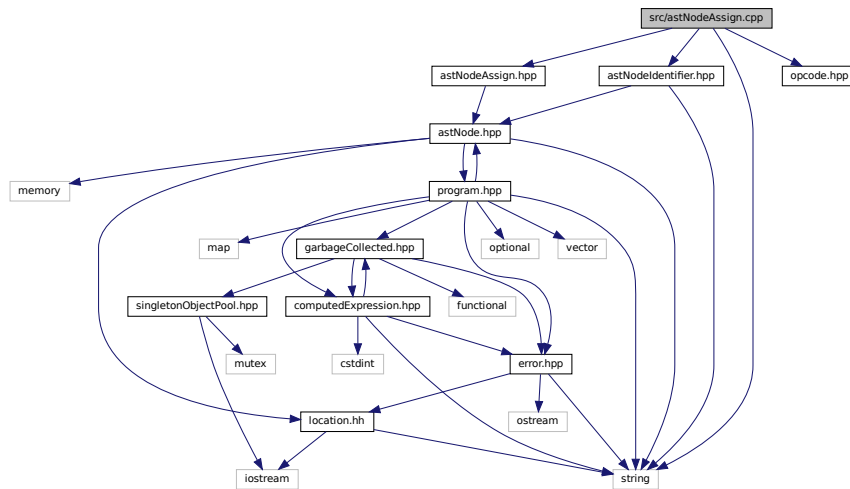
## 6.34 src/astNodeAssign.cpp File Reference

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

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

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

Include dependency graph for astNodeAssign.cpp:



### 6.34.1 Detailed Description

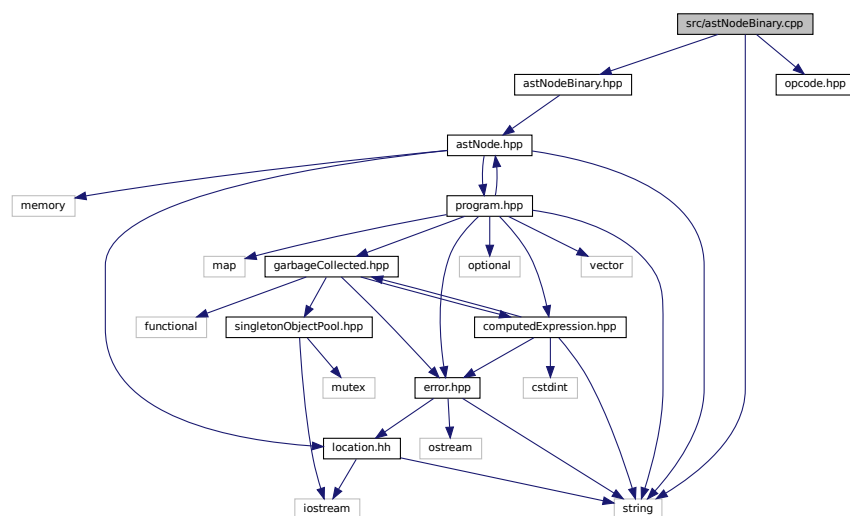
Define the [Tang::AstNodeAssign](#) class.

## 6.35 src/astNodeBinary.cpp File Reference

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

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

Include dependency graph for astNodeBinary.cpp:



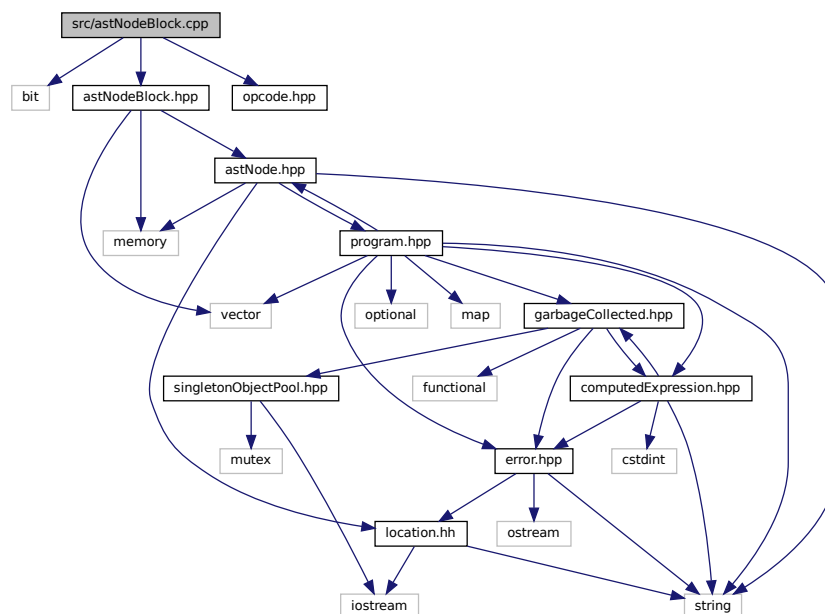
### 6.35.1 Detailed Description

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

## 6.36 src/astNodeBlock.cpp File Reference

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

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



### 6.36.1 Detailed Description

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

## 6.37 src/astNodeBoolean.cpp File Reference

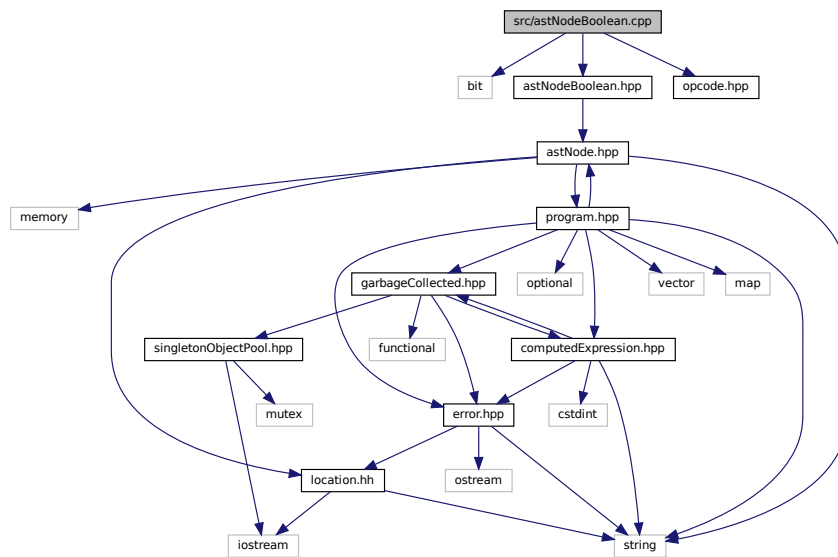
Define the [Tang::AstNodeBoolean](#) class.

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



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

Include dependency graph for astNodeBoolean.cpp:



### 6.37.1 Detailed Description

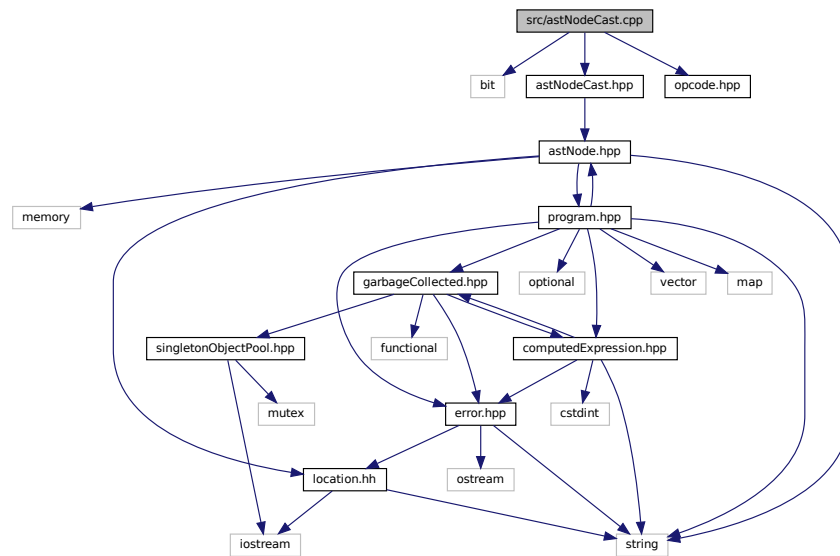
Define the [Tang::AstNodeBoolean](#) class.

## 6.38 src/astNodeCast.cpp File Reference

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

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

Include dependency graph for astNodeCast.cpp:



### 6.38.1 Detailed Description

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

## 6.39 src/astNodeDoWhile.cpp File Reference

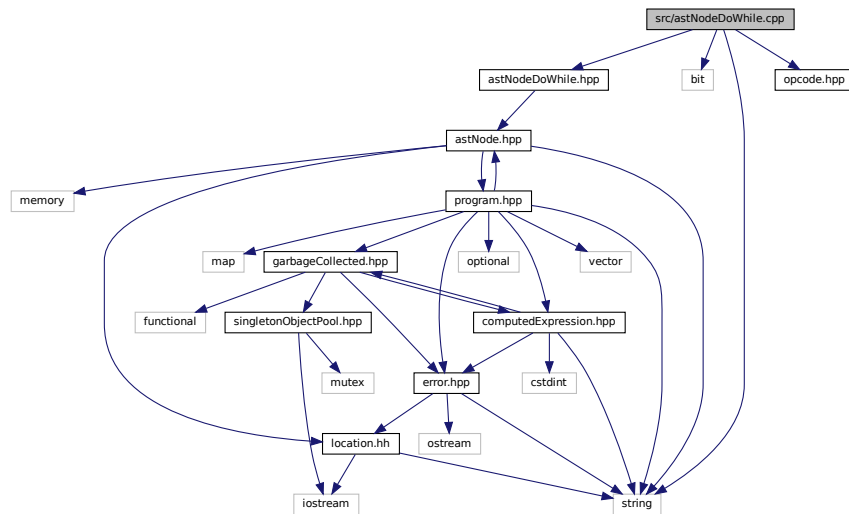
Define the [Tang::AstNodeDoWhile](#) class.

```

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

```

Include dependency graph for astNodeDoWhile.cpp:



### 6.39.1 Detailed Description

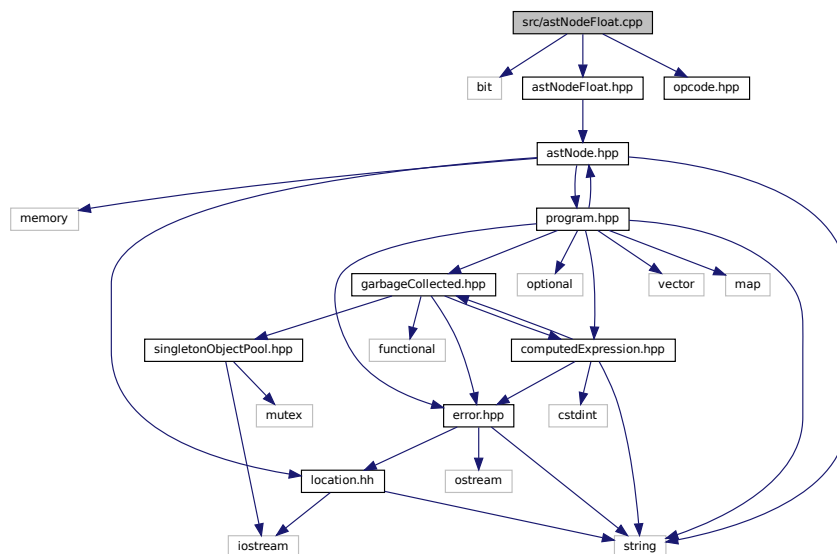
Define the [Tang::AstNodeDoWhile](#) class.

## 6.40 src/astNodeFloat.cpp File Reference

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

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

Include dependency graph for astNodeFloat.cpp:



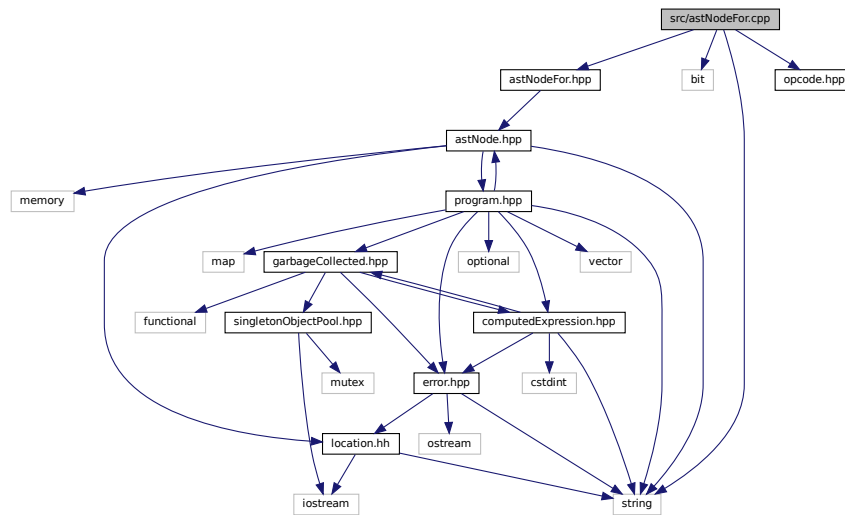
### 6.40.1 Detailed Description

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

## 6.41 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:
```



### 6.41.1 Detailed Description

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

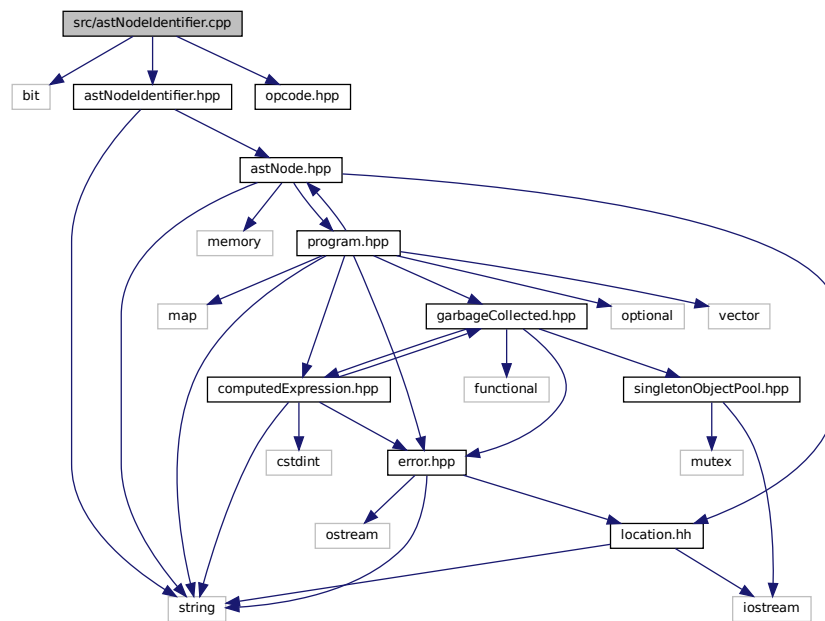
## 6.42 src/astNodeIdentifier.cpp File Reference

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

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

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

Include dependency graph for astNodeIdentifier.cpp:



### 6.42.1 Detailed Description

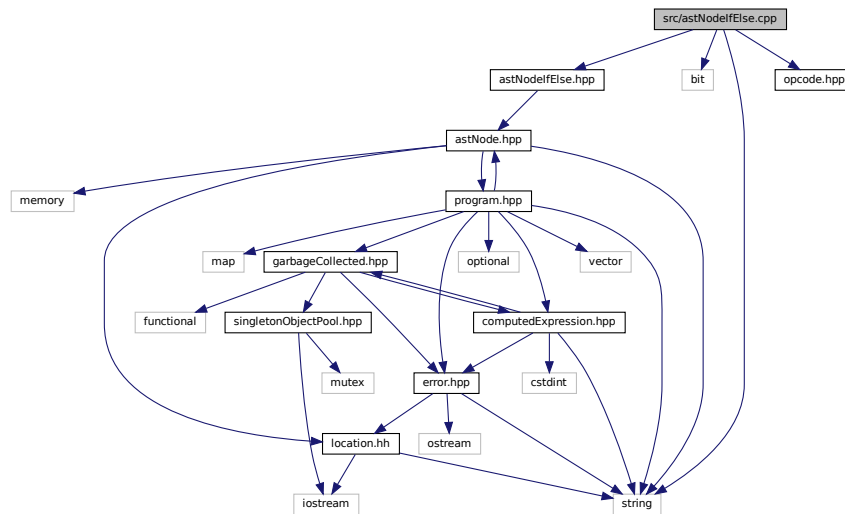
Define the [Tang::AstNodeIdentifier](#) class.

## 6.43 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`:



### 6.43.1 Detailed Description

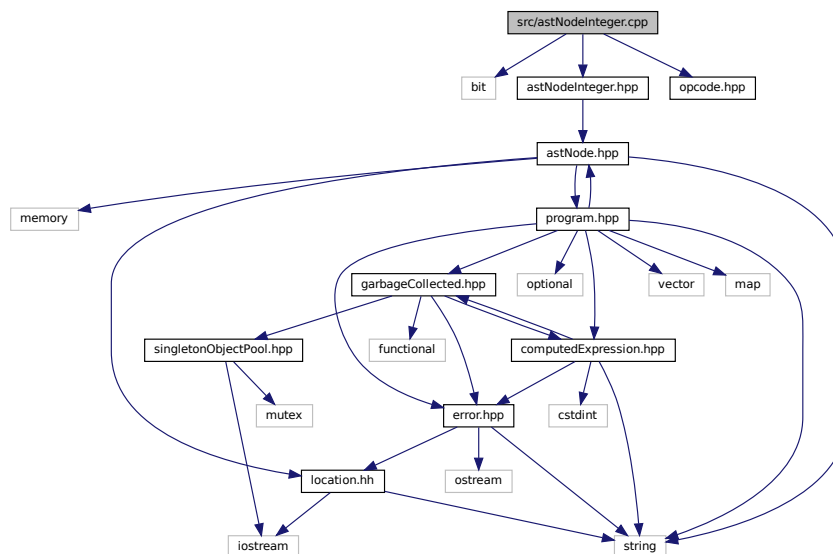
Define the [Tang::AstNodeIfElse](#) class.

## 6.44 src/astNodeInteger.cpp File Reference

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

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

Include dependency graph for `astNodeInteger.cpp`:



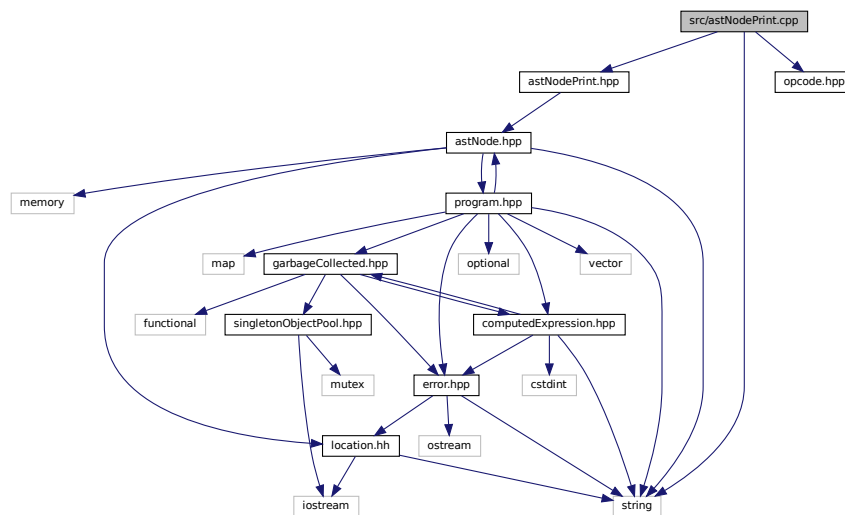
### 6.44.1 Detailed Description

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

## 6.45 src/astNodePrint.cpp File Reference

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

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



### 6.45.1 Detailed Description

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

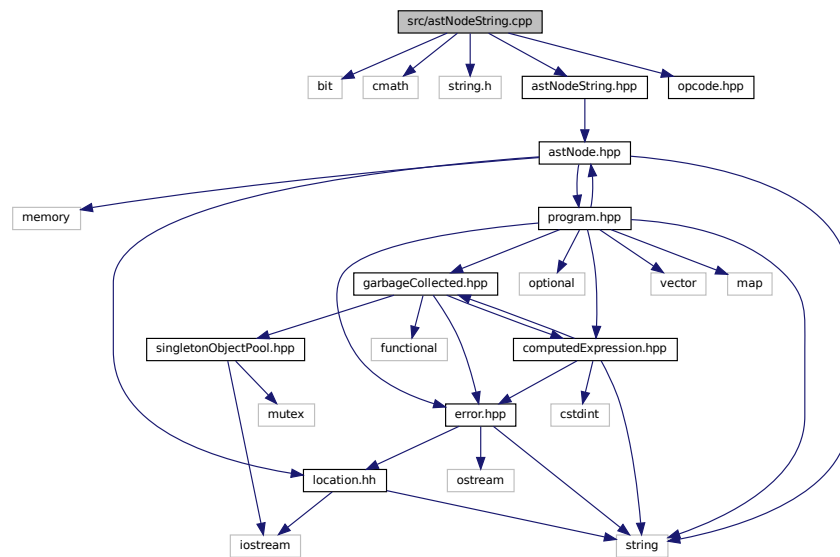
## 6.46 src/astNodeString.cpp File Reference

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

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

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

Include dependency graph for `astNodeString.cpp`:



### 6.46.1 Detailed Description

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

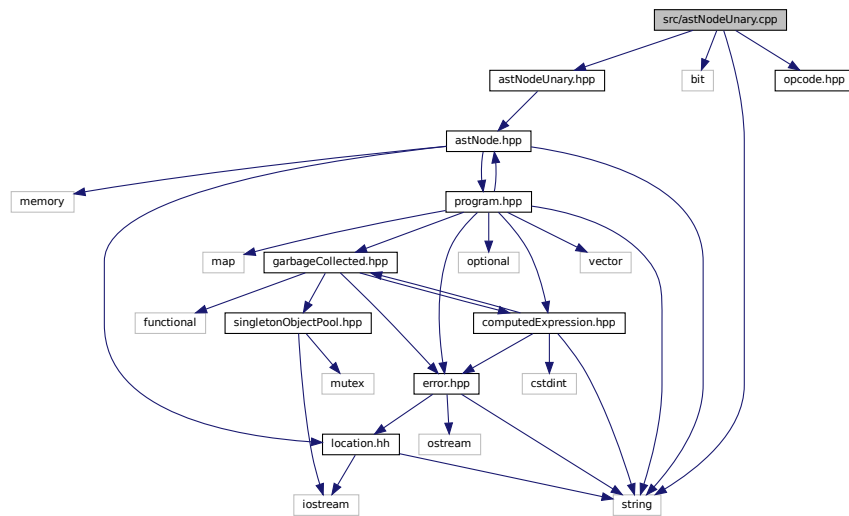
## 6.47 src/astNodeUnary.cpp File Reference

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

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



Include dependency graph for astNodeUnary.cpp:



### 6.47.1 Detailed Description

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

## 6.48 src/astNodeWhile.cpp File Reference

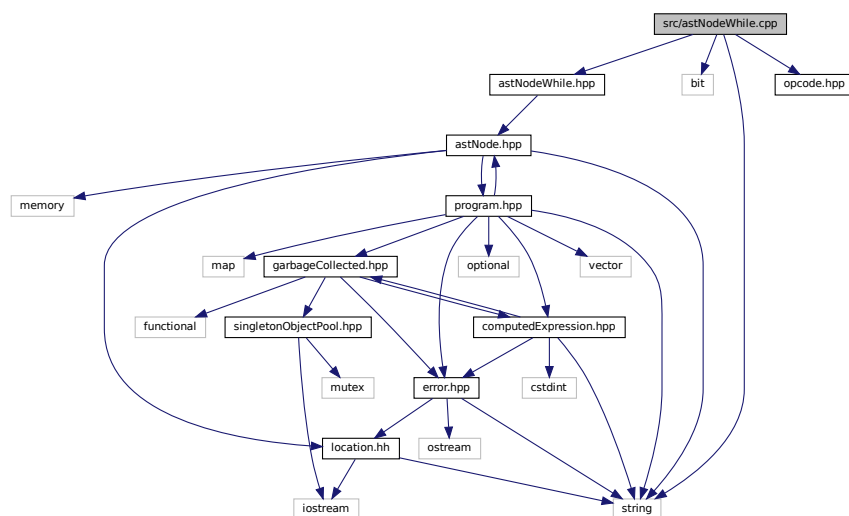
Define the [Tang::AstNodeWhile](#) class.

```

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

```

Include dependency graph for astNodeWhile.cpp:



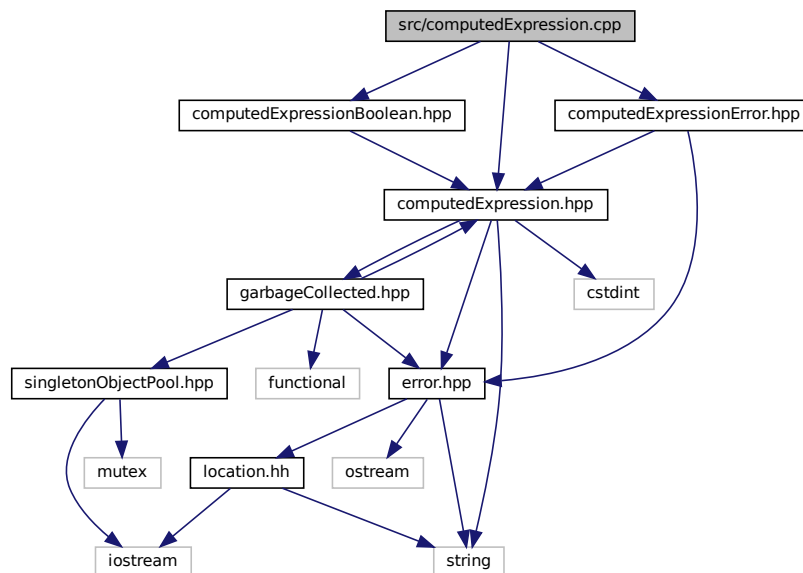
### 6.48.1 Detailed Description

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

## 6.49 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:
```



### 6.49.1 Detailed Description

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

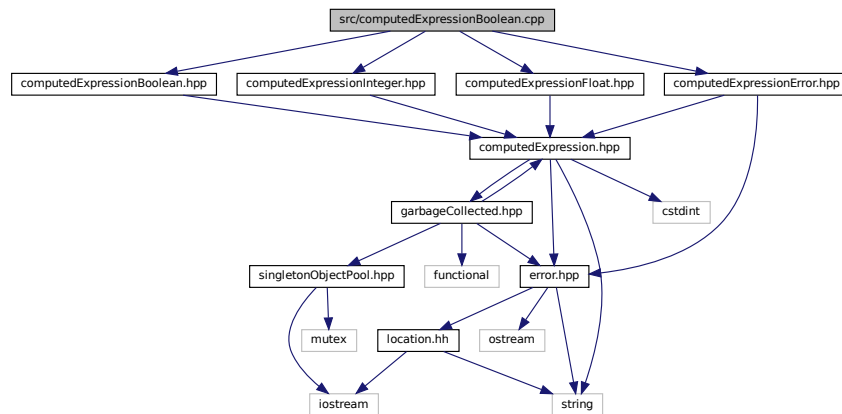
## 6.50 src/computedExpressionBoolean.cpp File Reference

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

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

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

Include dependency graph for computedExpressionBoolean.cpp:



### 6.50.1 Detailed Description

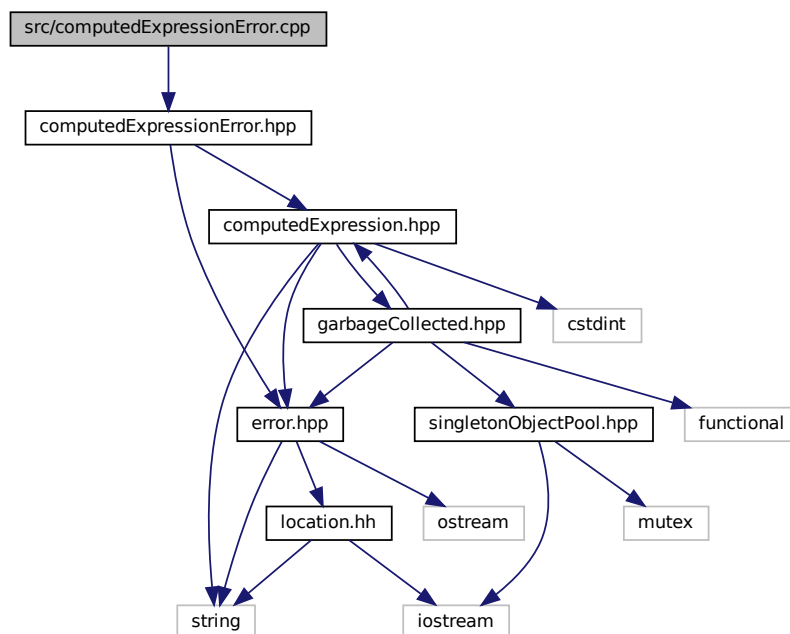
Define the [Tang::ComputedExpressionBoolean](#) class.

## 6.51 src/computedExpressionError.cpp File Reference

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

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

Include dependency graph for computedExpressionError.cpp:



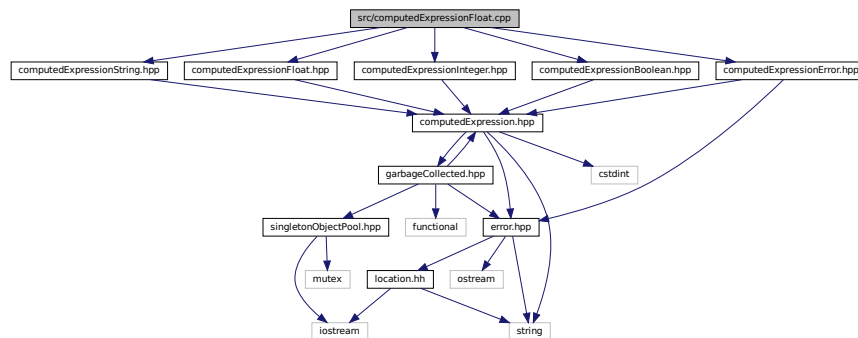
### 6.51.1 Detailed Description

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

## 6.52 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:
```



### 6.52.1 Detailed Description

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

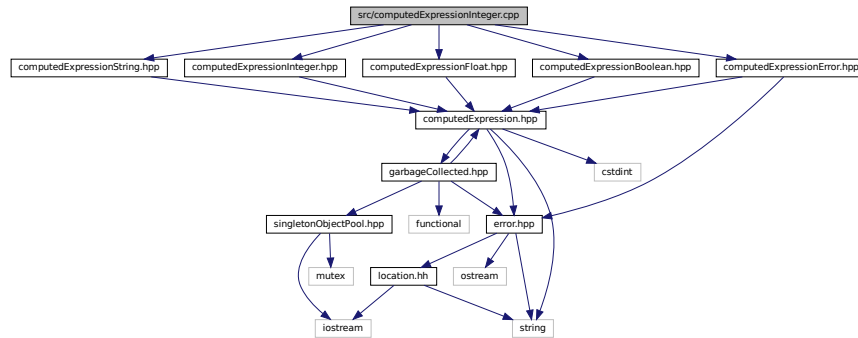
## 6.53 src/computedExpressionInteger.cpp File Reference

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:



### 6.53.1 Detailed Description

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

## 6.54 src/computedExpressionString.cpp File Reference

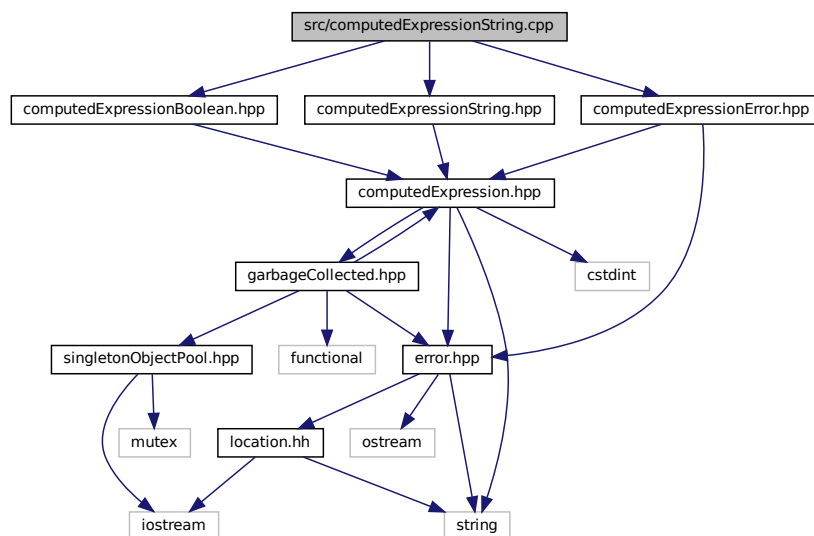
Define the [Tang::ComputedExpressionString](#) class.

```
#include "computedExpressionString.hpp"
```

```
#include "computedExpressionBoolean.hpp"
```

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

Include dependency graph for computedExpressionString.cpp:



### 6.54.1 Detailed Description

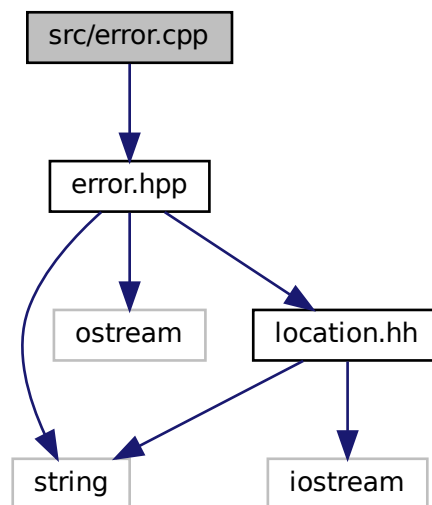
Define the [Tang::ComputedExpressionString](#) class.

## 6.55 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)`

### 6.55.1 Detailed Description

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

### 6.55.2 Function Documentation

#### 6.55.2.1 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

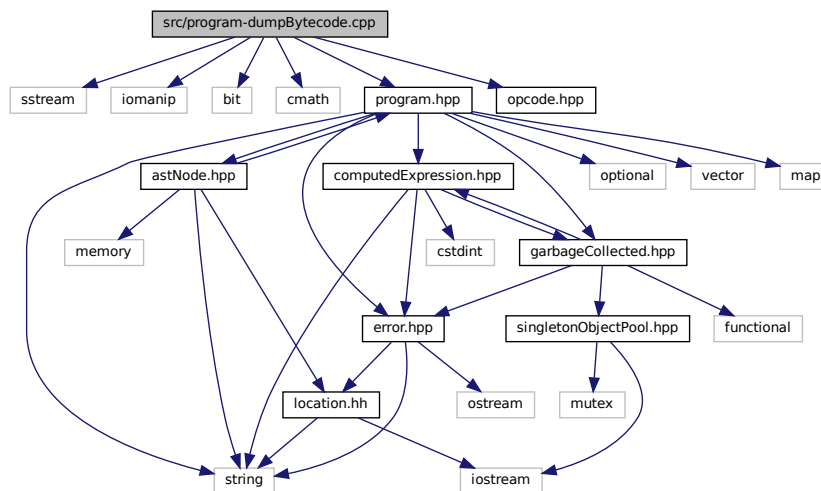
The output stream.

## 6.56 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.

#### 6.56.1 Detailed Description

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

## 6.56.2 Macro Definition Documentation

### 6.56.2.1 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

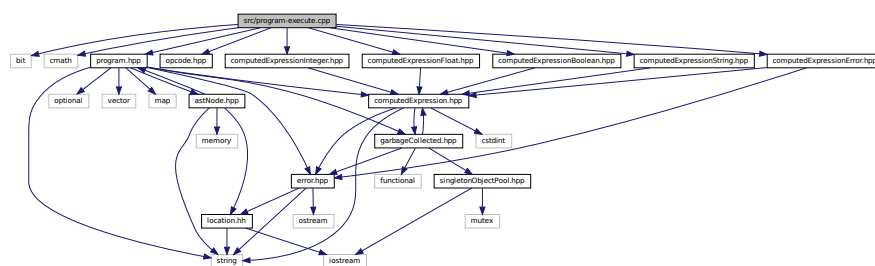
x	The number of additional vector entries that should exist.
---	--

## 6.57 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 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.

## 6.57.1 Detailed Description

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

## 6.57.2 Macro Definition Documentation

### 6.57.2.1 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

<code>x</code>	The number of additional vector entries that should exist.
----------------	--

### 6.57.2.2 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

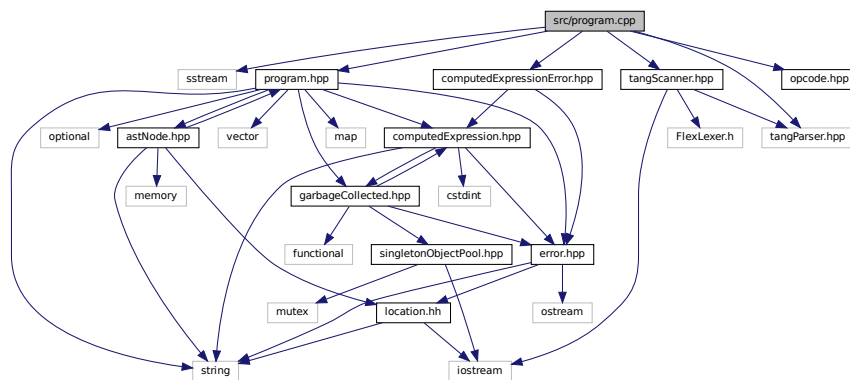
<code>x</code>	The number of entries that should exist in the stack.
----------------	---

## 6.58 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 "computedExpressionError.hpp"
```

Include dependency graph for program.cpp:



### 6.58.1 Detailed Description

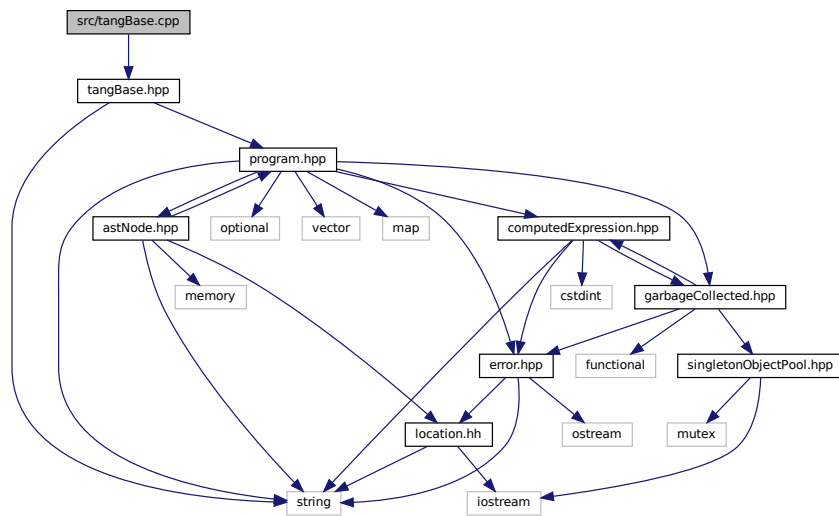
Define the [Tang::Program](#) class.

## 6.59 src/tangBase.cpp File Reference

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

```
#include "tangBase.hpp"
```

Include dependency graph for tangBase.cpp:



### 6.59.1 Detailed Description

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

## 6.60 test/test.cpp File Reference

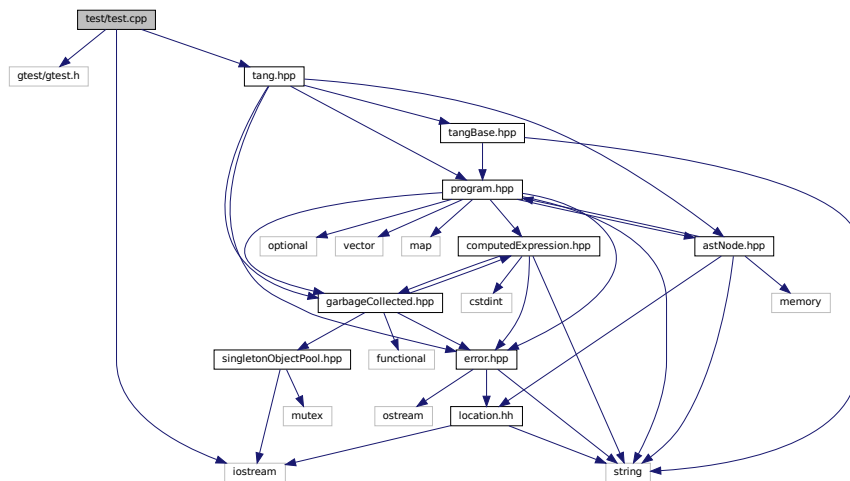
Test the general language behaviors.

```
#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** (CodeBlock, Statements)
- **TEST** (Assign, Identifier)
- **TEST** (ControlFlow, IfElse)
- **TEST** (ControlFlow, While)
- **TEST** (ControlFlow, DoWhile)
- **TEST** (ControlFlow, For)
- **TEST** (Print, Default)
- **int main** (int argc, char \*\*argv)

### 6.60.1 Detailed Description

Test the general language behaviors.

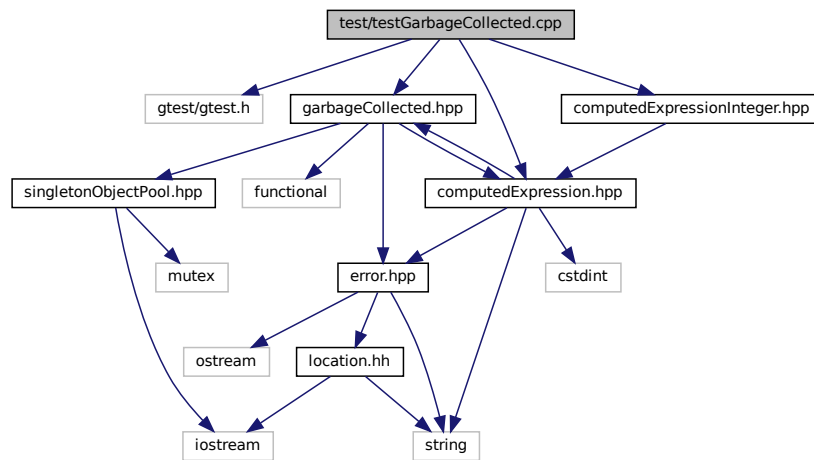
## 6.61 test/testGarbageCollected.cpp File Reference

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)

### 6.61.1 Detailed Description

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

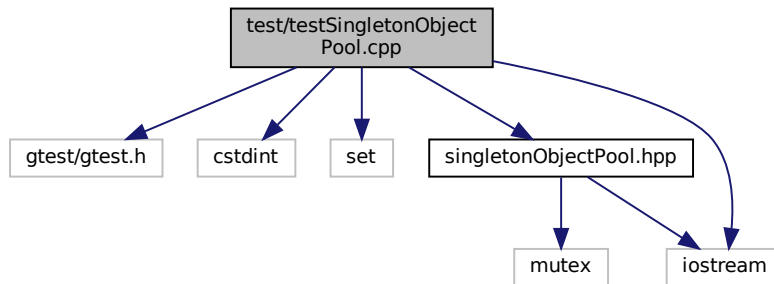
## 6.62 test/testSingletonObjectPool.cpp File Reference

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, RecycledObjectsIsReused)
- **TEST** (Get, SuccessiveCallsAreSequential)
- **TEST** (Get, KeepsGeneratingDifferentPointers)
- **TEST** (Recycle, WorksAfterLargeNumberOfAllocations)
- `int main` (`int argc`, `char **argv`)

### 6.62.1 Detailed Description

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

# Index

- \_\_add
    - Tang::ComputedExpression, [48](#)
    - Tang::ComputedExpressionBoolean, [59](#)
    - Tang::ComputedExpressionError, [68](#)
    - Tang::ComputedExpressionFloat, [78](#)
    - Tang::ComputedExpressionInteger, [88](#)
    - Tang::ComputedExpressionString, [98](#)
  - \_\_boolean
    - Tang::ComputedExpression, [49](#)
    - Tang::ComputedExpressionBoolean, [59](#)
    - Tang::ComputedExpressionError, [69](#)
    - Tang::ComputedExpressionFloat, [78](#)
    - Tang::ComputedExpressionInteger, [88](#)
    - Tang::ComputedExpressionString, [98](#)
  - \_\_divide
    - Tang::ComputedExpression, [49](#)
    - Tang::ComputedExpressionBoolean, [59](#)
    - Tang::ComputedExpressionError, [69](#)
    - Tang::ComputedExpressionFloat, [78](#)
    - Tang::ComputedExpressionInteger, [88](#)
    - Tang::ComputedExpressionString, [98](#)
  - \_\_equal
    - Tang::ComputedExpression, [49](#)
    - Tang::ComputedExpressionBoolean, [60](#)
    - Tang::ComputedExpressionError, [69](#)
    - Tang::ComputedExpressionFloat, [79](#)
    - Tang::ComputedExpressionInteger, [89](#)
    - Tang::ComputedExpressionString, [99](#)
  - \_\_float
    - Tang::ComputedExpression, [50](#)
    - Tang::ComputedExpressionBoolean, [60](#)
    - Tang::ComputedExpressionError, [70](#)
    - Tang::ComputedExpressionFloat, [79](#)
    - Tang::ComputedExpressionInteger, [89](#)
    - Tang::ComputedExpressionString, [99](#)
  - \_\_integer
    - Tang::ComputedExpression, [50](#)
    - Tang::ComputedExpressionBoolean, [60](#)
    - Tang::ComputedExpressionError, [70](#)
    - Tang::ComputedExpressionFloat, [79](#)
    - Tang::ComputedExpressionInteger, [89](#)
    - Tang::ComputedExpressionString, [99](#)
  - \_\_lessThan
    - Tang::ComputedExpression, [50](#)
    - Tang::ComputedExpressionBoolean, [61](#)
    - Tang::ComputedExpressionError, [70](#)
    - Tang::ComputedExpressionFloat, [80](#)
    - Tang::ComputedExpressionInteger, [90](#)
    - Tang::ComputedExpressionString, [99](#)
  - \_\_modulo
    - Tang::ComputedExpression, [51](#)
    - Tang::ComputedExpressionBoolean, [61](#)
    - Tang::ComputedExpressionError, [71](#)
    - Tang::ComputedExpressionFloat, [80](#)
    - Tang::ComputedExpressionInteger, [90](#)
    - Tang::ComputedExpressionString, [100](#)
  - \_\_multiply
    - Tang::ComputedExpression, [51](#)
    - Tang::ComputedExpressionBoolean, [62](#)
    - Tang::ComputedExpressionError, [71](#)
    - Tang::ComputedExpressionFloat, [80](#)
    - Tang::ComputedExpressionInteger, [90](#)
    - Tang::ComputedExpressionString, [100](#)
  - \_\_negative
    - Tang::ComputedExpression, [51](#)
    - Tang::ComputedExpressionBoolean, [62](#)
    - Tang::ComputedExpressionError, [71](#)
    - Tang::ComputedExpressionFloat, [81](#)
    - Tang::ComputedExpressionInteger, [91](#)
    - Tang::ComputedExpressionString, [101](#)
  - \_\_not
    - Tang::ComputedExpression, [52](#)
    - Tang::ComputedExpressionBoolean, [62](#)
    - Tang::ComputedExpressionError, [72](#)
    - Tang::ComputedExpressionFloat, [81](#)
    - Tang::ComputedExpressionInteger, [91](#)
    - Tang::ComputedExpressionString, [101](#)
  - \_\_string
    - Tang::ComputedExpression, [52](#)
    - Tang::ComputedExpressionBoolean, [62](#)
    - Tang::ComputedExpressionError, [72](#)
    - Tang::ComputedExpressionFloat, [81](#)
    - Tang::ComputedExpressionInteger, [91](#)
    - Tang::ComputedExpressionString, [101](#)
  - \_\_subtract
    - Tang::ComputedExpression, [52](#)
    - Tang::ComputedExpressionBoolean, [63](#)
    - Tang::ComputedExpressionError, [72](#)
    - Tang::ComputedExpressionFloat, [82](#)
    - Tang::ComputedExpressionInteger, [92](#)
    - Tang::ComputedExpressionString, [101](#)
- ~GarbageCollected
  - Tang::GarbageCollected, [110](#)
- ADD
  - opcode.hpp, [162](#)
- Add
  - Tang::AstNodeBinary, [17](#)
- addBytecode

- Tang::Program, [128](#)
- AstNode
  - Tang::AstNode, [13](#)
- AstNodeAssign
  - Tang::AstNodeAssign, [15](#)
- AstNodeBinary
  - Tang::AstNodeBinary, [18](#)
- AstNodeBlock
  - Tang::AstNodeBlock, [20](#)
- AstNodeBoolean
  - Tang::AstNodeBoolean, [21](#)
- AstNodeCast
  - Tang::AstNodeCast, [24](#)
- AstNodeDoWhile
  - Tang::AstNodeDoWhile, [26](#)
- AstNodeFloat
  - Tang::AstNodeFloat, [28](#)
- AstNodeFor
  - Tang::AstNodeFor, [30](#)
- AstNodeIdentifier
  - Tang::AstNodeIdentifier, [32](#)
- AstNodeIfElse
  - Tang::AstNodeIfElse, [34](#)
- AstNodeInteger
  - Tang::AstNodeInteger, [37](#)
- AstNodePrint
  - Tang::AstNodePrint, [40](#)
- AstNodeString
  - Tang::AstNodeString, [42](#)
- AstNodeUnary
  - Tang::AstNodeUnary, [44](#)
- AstNodeWhile
  - Tang::AstNodeWhile, [46](#)
- BOOLEAN
  - opcode.hpp, [162](#)
- Boolean
  - Tang::AstNodeCast, [24](#)
- build/generated/location.hh, [135](#)
- CASTBOOLEAN
  - opcode.hpp, [162](#)
- CASTFLOAT
  - opcode.hpp, [162](#)
- CASTINTEGER
  - opcode.hpp, [162](#)
- CodeType
  - Tang::Program, [127](#)
- collectIdentifiers
  - Tang::AstNode, [13](#)
  - Tang::AstNodeAssign, [15](#)
  - Tang::AstNodeBinary, [18](#)
  - Tang::AstNodeBlock, [20](#)
  - Tang::AstNodeBoolean, [22](#)
  - Tang::AstNodeCast, [24](#)
  - Tang::AstNodeDoWhile, [26](#)
  - Tang::AstNodeFloat, [28](#)
  - Tang::AstNodeFor, [30](#)
  - Tang::AstNodeIdentifier, [32](#)
  - Tang::AstNodeIfElse, [36](#)
  - Tang::AstNodeInteger, [38](#)
  - Tang::AstNodePrint, [40](#)
  - Tang::AstNodeString, [42](#)
  - Tang::AstNodeUnary, [44](#)
  - Tang::AstNodeWhile, [46](#)
- compileScript
  - Tang::TangBase, [132](#)
- ComputedExpressionBoolean
  - Tang::ComputedExpressionBoolean, [58](#)
- ComputedExpressionError
  - Tang::ComputedExpressionError, [68](#)
- ComputedExpressionFloat
  - Tang::ComputedExpressionFloat, [77](#)
- ComputedExpressionInteger
  - Tang::ComputedExpressionInteger, [87](#)
- ComputedExpressionString
  - Tang::ComputedExpressionString, [97](#)
- Default
  - Tang::AstNodePrint, [40](#)
- DIVIDE
  - opcode.hpp, [162](#)
- Divide
  - Tang::AstNodeBinary, [17](#)
- dump
  - Tang::ComputedExpression, [53](#)
  - Tang::ComputedExpressionBoolean, [63](#)
  - Tang::ComputedExpressionError, [73](#)
  - Tang::ComputedExpressionFloat, [82](#)
  - Tang::ComputedExpressionInteger, [92](#)
  - Tang::ComputedExpressionString, [102](#)
- dumpBytecode
  - Tang::Program, [128](#)
- DUMPPROGRAMCHECK
  - program-dumpBytecode.cpp, [186](#)
- EQ
  - opcode.hpp, [162](#)
- Equal
  - Tang::AstNodeBinary, [17](#)
- Error
  - Tang::Error, [106](#)
- error.cpp
  - operator<<, [184](#)
- execute
  - Tang::Program, [128](#)
- EXECUTEPROGRAMCHECK
  - program-execute.cpp, [187](#)
- FLOAT
  - opcode.hpp, [162](#)
- Float
  - Tang::AstNodeCast, [24](#)
- GarbageCollected
  - Tang::GarbageCollected, [109](#), [110](#)
- get
  - Tang::SingletonObjectPool< T >, [130](#)



- get\_next\_token
  - Tang::TangScanner, 134
- getAst
  - Tang::Program, 128
- getBytecode
  - Tang::Program, 129
- getCode
  - Tang::Program, 129
- getInstance
  - Tang::SingletonObjectPool< T >, 131
- getResult
  - Tang::Program, 129
- GreaterThan
  - Tang::AstNodeBinary, 17
- GreaterThanEqual
  - Tang::AstNodeBinary, 17
- GT
  - opcode.hpp, 162
- GTE
  - opcode.hpp, 162
- include/astNode.hpp, 137
- include/astNodeAssign.hpp, 138
- include/astNodeBinary.hpp, 139
- include/astNodeBlock.hpp, 140
- include/astNodeBoolean.hpp, 141
- include/astNodeCast.hpp, 142
- include/astNodeDoWhile.hpp, 143
- include/astNodeFloat.hpp, 144
- include/astNodeFor.hpp, 145
- include/astNodeIdentifier.hpp, 146
- include/astNodeIfElse.hpp, 147
- include/astNodeInteger.hpp, 148
- include/astNodePrint.hpp, 149
- include/astNodeString.hpp, 150
- include/astNodeUnary.hpp, 151
- include/astNodeWhile.hpp, 152
- include/computedExpression.hpp, 153
- include/computedExpressionBoolean.hpp, 154
- include/computedExpressionError.hpp, 155
- include/computedExpressionFloat.hpp, 156
- include/computedExpressionInteger.hpp, 157
- include/computedExpressionString.hpp, 158
- include/error.hpp, 159
- include/garbageCollected.hpp, 160
- include/macros.hpp, 160
- include/opcode.hpp, 161
- include/program.hpp, 163
- include/singletonObjectPool.hpp, 164
- include/tang.hpp, 165
- include/tangBase.hpp, 166
- include/tangScanner.hpp, 167
- INTEGER
  - opcode.hpp, 162
- Integer
  - Tang::AstNodeCast, 24
- is\_equal
  - Tang::ComputedExpression, 53, 55, 56
  - Tang::ComputedExpressionBoolean, 63–65
  - Tang::ComputedExpressionError, 73–75
  - Tang::ComputedExpressionFloat, 83, 84
  - Tang::ComputedExpressionInteger, 93, 94
  - Tang::ComputedExpressionString, 102–104
- JMP
  - opcode.hpp, 162
- JMPF\_POP
  - opcode.hpp, 162
- JMPT\_POP
  - opcode.hpp, 162
- LessThan
  - Tang::AstNodeBinary, 17
- LessThanEqual
  - Tang::AstNodeBinary, 17
- location.hh
  - operator<<, 136, 137
- LT
  - opcode.hpp, 162
- LTE
  - opcode.hpp, 162
- macros.hpp
  - TANG\_UNUSED, 161
- make
  - Tang::GarbageCollected, 110
- makeCopy
  - Tang::ComputedExpression, 56
  - Tang::ComputedExpressionBoolean, 66
  - Tang::ComputedExpressionError, 75
  - Tang::ComputedExpressionFloat, 85
  - Tang::ComputedExpressionInteger, 95
  - Tang::ComputedExpressionString, 104
- MODULO
  - opcode.hpp, 162
- Modulo
  - Tang::AstNodeBinary, 17
- MULTIPLY
  - opcode.hpp, 162
- Multiply
  - Tang::AstNodeBinary, 17
- NEGATIVE
  - opcode.hpp, 162
- Negative
  - Tang::AstNodeUnary, 44
- NEQ
  - opcode.hpp, 162
- NOT
  - opcode.hpp, 162
- Not
  - Tang::AstNodeUnary, 44
- NotEqual
  - Tang::AstNodeBinary, 17
- NULLVAL
  - opcode.hpp, 162
- Opcode

- opcode.hpp, 162
- opcode.hpp
  - ADD, 162
  - BOOLEAN, 162
  - CASTBOOLEAN, 162
  - CASTFLOAT, 162
  - CASTINTEGER, 162
  - DIVIDE, 162
  - EQ, 162
  - FLOAT, 162
  - GT, 162
  - GTE, 162
  - INTEGER, 162
  - JMP, 162
  - JMPF\_POP, 162
  - JMPT\_POP, 162
  - LT, 162
  - LTE, 162
  - MODULO, 162
  - MULTIPLY, 162
  - NEGATIVE, 162
  - NEQ, 162
  - NOT, 162
  - NULLVAL, 162
  - Opcodes, 162
  - PEEK, 162
  - POKE, 162
  - POP, 162
  - PRINT, 162
  - STRING, 162
  - SUBTRACT, 162
- Operation
  - Tang::AstNodeBinary, 17
- Operator
  - Tang::AstNodeUnary, 44
- operator!
  - Tang::GarbageCollected, 111
- operator!=
  - Tang::GarbageCollected, 111
- operator<
  - Tang::GarbageCollected, 116
- operator<<
  - error.cpp, 184
  - location.hh, 136, 137
  - Tang::Error, 106
  - Tang::GarbageCollected, 122
- operator<=
  - Tang::GarbageCollected, 116
- operator>
  - Tang::GarbageCollected, 121
- operator>=
  - Tang::GarbageCollected, 121
- operator\*
  - Tang::GarbageCollected, 112, 113
- operator+
  - Tang::GarbageCollected, 113
- operator-
  - Tang::GarbageCollected, 114
- operator->
  - Tang::GarbageCollected, 115
- operator/
  - Tang::GarbageCollected, 115
- operator=
  - Tang::GarbageCollected, 117
- operator==
  - Tang::GarbageCollected, 118–120
- operator%
  - Tang::GarbageCollected, 112
- PEEK
  - opcode.hpp, 162
- POKE
  - opcode.hpp, 162
- POP
  - opcode.hpp, 162
- PRINT
  - opcode.hpp, 162
- Program
  - Tang::Program, 127
- program-dumpBytecode.cpp
  - DUMPPROGRAMCHECK, 186
- program-execute.cpp
  - EXECUTEPROGRAMCHECK, 187
  - STACKCHECK, 187
- recycle
  - Tang::SingletonObjectPool< T >, 131
- Script
  - Tang::Program, 127
- setJumpTarget
  - Tang::Program, 129
- src/astNode.cpp, 168
- src/astNodeAssign.cpp, 168
- src/astNodeBinary.cpp, 169
- src/astNodeBlock.cpp, 170
- src/astNodeBoolean.cpp, 170
- src/astNodeCast.cpp, 171
- src/astNodeDoWhile.cpp, 172
- src/astNodeFloat.cpp, 173
- src/astNodeFor.cpp, 174
- src/astNodeIdentifier.cpp, 174
- src/astNodeIfElse.cpp, 175
- src/astNodeInteger.cpp, 176
- src/astNodePrint.cpp, 177
- src/astNodeString.cpp, 177
- src/astNodeUnary.cpp, 178
- src/astNodeWhile.cpp, 179
- src/computedExpression.cpp, 180
- src/computedExpressionBoolean.cpp, 180
- src/computedExpressionError.cpp, 181
- src/computedExpressionFloat.cpp, 182
- src/computedExpressionInteger.cpp, 182
- src/computedExpressionString.cpp, 183
- src/error.cpp, 184
- src/program-dumpBytecode.cpp, 185
- src/program-execute.cpp, 186

- src/program.cpp, 188
- src/tangBase.cpp, 188
- STACKCHECK
  - program-execute.cpp, 187
- STRING
  - opcode.hpp, 162
- SUBTRACT
  - opcode.hpp, 162
- Subtract
  - Tang::AstNodeBinary, 17
- Tang::AstNode, 11
  - AstNode, 13
  - collectIdentifiers, 13
- Tang::AstNodeAssign, 14
  - AstNodeAssign, 15
  - collectIdentifiers, 15
- Tang::AstNodeBinary, 16
  - Add, 17
  - AstNodeBinary, 18
  - collectIdentifiers, 18
  - Divide, 17
  - Equal, 17
  - GreaterThan, 17
  - GreaterThanEqual, 17
  - LessThan, 17
  - LessThanEqual, 17
  - Modulo, 17
  - Multiply, 17
  - NotEqual, 17
  - Operation, 17
  - Subtract, 17
- Tang::AstNodeBlock, 18
  - AstNodeBlock, 20
  - collectIdentifiers, 20
- Tang::AstNodeBoolean, 20
  - AstNodeBoolean, 21
  - collectIdentifiers, 22
- Tang::AstNodeCast, 22
  - AstNodeCast, 24
  - Boolean, 24
  - collectIdentifiers, 24
  - Float, 24
  - Integer, 24
  - Type, 24
- Tang::AstNodeDoWhile, 25
  - AstNodeDoWhile, 26
  - collectIdentifiers, 26
- Tang::AstNodeFloat, 27
  - AstNodeFloat, 28
  - collectIdentifiers, 28
- Tang::AstNodeFor, 29
  - AstNodeFor, 30
  - collectIdentifiers, 30
- Tang::AstNodeIdentifier, 31
  - AstNodeIdentifier, 32
  - collectIdentifiers, 32
- Tang::AstNodeIfElse, 33
  - AstNodeIfElse, 34
- collectIdentifiers, 36
- Tang::AstNodeInteger, 36
  - AstNodeInteger, 37
  - collectIdentifiers, 38
- Tang::AstNodePrint, 38
  - AstNodePrint, 40
  - collectIdentifiers, 40
  - Default, 40
  - Type, 39
- Tang::AstNodeString, 41
  - AstNodeString, 42
  - collectIdentifiers, 42
- Tang::AstNodeUnary, 42
  - AstNodeUnary, 44
  - collectIdentifiers, 44
  - Negative, 44
  - Not, 44
  - Operator, 44
- Tang::AstNodeWhile, 45
  - AstNodeWhile, 46
  - collectIdentifiers, 46
- Tang::ComputedExpression, 47
  - \_\_add, 48
  - \_\_boolean, 49
  - \_\_divide, 49
  - \_\_equal, 49
  - \_\_float, 50
  - \_\_integer, 50
  - \_\_lessThan, 50
  - \_\_modulo, 51
  - \_\_multiply, 51
  - \_\_negative, 51
  - \_\_not, 52
  - \_\_string, 52
  - \_\_subtract, 52
  - dump, 53
  - is\_equal, 53, 55, 56
  - makeCopy, 56
- Tang::ComputedExpressionBoolean, 57
  - \_\_add, 59
  - \_\_boolean, 59
  - \_\_divide, 59
  - \_\_equal, 60
  - \_\_float, 60
  - \_\_integer, 60
  - \_\_lessThan, 61
  - \_\_modulo, 61
  - \_\_multiply, 62
  - \_\_negative, 62
  - \_\_not, 62
  - \_\_string, 62
  - \_\_subtract, 63
  - ComputedExpressionBoolean, 58
  - dump, 63
  - is\_equal, 63–65
  - makeCopy, 66
- Tang::ComputedExpressionError, 66
  - \_\_add, 68

- \_\_boolean, 69
- \_\_divide, 69
- \_\_equal, 69
- \_\_float, 70
- \_\_integer, 70
- \_\_lessThan, 70
- \_\_modulo, 71
- \_\_multiply, 71
- \_\_negative, 71
- \_\_not, 72
- \_\_string, 72
- \_\_subtract, 72
- ComputedExpressionError, 68
- dump, 73
- is\_equal, 73–75
- makeCopy, 75
- Tang::ComputedExpressionFloat, 76
  - \_\_add, 78
  - \_\_boolean, 78
  - \_\_divide, 78
  - \_\_equal, 79
  - \_\_float, 79
  - \_\_integer, 79
  - \_\_lessThan, 80
  - \_\_modulo, 80
  - \_\_multiply, 80
  - \_\_negative, 81
  - \_\_not, 81
  - \_\_string, 81
  - \_\_subtract, 82
  - ComputedExpressionFloat, 77
  - dump, 82
  - is\_equal, 83, 84
  - makeCopy, 85
- Tang::ComputedExpressionInteger, 85
  - \_\_add, 88
  - \_\_boolean, 88
  - \_\_divide, 88
  - \_\_equal, 89
  - \_\_float, 89
  - \_\_integer, 89
  - \_\_lessThan, 90
  - \_\_modulo, 90
  - \_\_multiply, 90
  - \_\_negative, 91
  - \_\_not, 91
  - \_\_string, 91
  - \_\_subtract, 92
  - ComputedExpressionInteger, 87
  - dump, 92
  - is\_equal, 93, 94
  - makeCopy, 95
- Tang::ComputedExpressionString, 95
  - \_\_add, 98
  - \_\_boolean, 98
  - \_\_divide, 98
  - \_\_equal, 99
  - \_\_float, 99
  - \_\_integer, 99
  - \_\_lessThan, 99
  - \_\_modulo, 100
  - \_\_multiply, 100
  - \_\_negative, 101
  - \_\_not, 101
  - \_\_string, 101
  - \_\_subtract, 101
  - ComputedExpressionString, 97
  - dump, 102
  - is\_equal, 102–104
  - makeCopy, 104
- Tang::Error, 105
  - Error, 106
  - operator<<, 106
- Tang::GarbageCollected, 107
  - ~GarbageCollected, 110
  - GarbageCollected, 109, 110
  - make, 110
  - operator!, 111
  - operator!=, 111
  - operator<, 116
  - operator<<, 122
  - operator<=, 116
  - operator>, 121
  - operator>=, 121
  - operator\*, 112, 113
  - operator+, 113
  - operator-, 114
  - operator->, 115
  - operator/, 115
  - operator=, 117
  - operator==, 118–120
  - operator%, 112
- Tang::location, 122
- Tang::position, 124
- Tang::Program, 125
  - addBytecode, 128
  - CodeType, 127
  - dumpBytecode, 128
  - execute, 128
  - getAst, 128
  - getBytecode, 129
  - getCode, 129
  - getResult, 129
  - Program, 127
  - Script, 127
  - setJumpTarget, 129
  - Template, 127
- Tang::SingletonObjectPool< T >, 130
  - get, 130
  - getInstance, 131
  - recycle, 131
- Tang::TangBase, 131
  - compileScript, 132
  - TangBase, 132
- Tang::TangScanner, 133
  - get\_next\_token, 134

- TangScanner, [134](#)
- TANG\_UNUSED
  - macros.hpp, [161](#)
- TangBase
  - Tang::TangBase, [132](#)
- TangScanner
  - Tang::TangScanner, [134](#)
- Template
  - Tang::Program, [127](#)
- test/test.cpp, [189](#)
- test/testGarbageCollected.cpp, [190](#)
- test/testSingletonObjectPool.cpp, [191](#)
- Type
  - Tang::AstNodeCast, [24](#)
  - Tang::AstNodePrint, [39](#)