

Tang

0.1

Generated by Doxygen 1.9.1

1 Tang: A Template Language	1
1.1 Quick Description	1
1.2 Features	1
1.3 License	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Class Documentation	11
5.1 Tang::AstNode Class Reference	11
5.1.1 Detailed Description	13
5.1.2 Member Enumeration Documentation	13
5.1.2.1 PreprocessState	13
5.1.3 Constructor & Destructor Documentation	13
5.1.3.1 AstNode()	13
5.1.4 Member Function Documentation	14
5.1.4.1 compile()	14
5.1.4.2 compilePreprocess()	14
5.1.4.3 dump()	15
5.2 Tang::AstNodeArray Class Reference	16
5.2.1 Detailed Description	17
5.2.2 Member Enumeration Documentation	17
5.2.2.1 PreprocessState	17
5.2.3 Constructor & Destructor Documentation	17
5.2.3.1 AstNodeArray()	17
5.2.4 Member Function Documentation	17
5.2.4.1 compile()	17
5.2.4.2 compilePreprocess()	19
5.2.4.3 dump()	19
5.3 Tang::AstNodeAssign Class Reference	20
5.3.1 Detailed Description	21
5.3.2 Member Enumeration Documentation	21
5.3.2.1 PreprocessState	21
5.3.3 Constructor & Destructor Documentation	21
5.3.3.1 AstNodeAssign()	21
5.3.4 Member Function Documentation	22
5.3.4.1 compile()	22
5.3.4.2 compilePreprocess()	22

5.3.4.3 dump()	23
5.4 Tang::AstNodeBinary Class Reference	23
5.4.1 Detailed Description	24
5.4.2 Member Enumeration Documentation	24
5.4.2.1 Operation	24
5.4.2.2 PreprocessState	25
5.4.3 Constructor & Destructor Documentation	25
5.4.3.1 AstNodeBinary()	25
5.4.4 Member Function Documentation	26
5.4.4.1 compile()	26
5.4.4.2 compilePreprocess()	26
5.4.4.3 dump()	27
5.5 Tang::AstNodeBlock Class Reference	27
5.5.1 Detailed Description	28
5.5.2 Member Enumeration Documentation	28
5.5.2.1 PreprocessState	28
5.5.3 Constructor & Destructor Documentation	29
5.5.3.1 AstNodeBlock()	29
5.5.4 Member Function Documentation	29
5.5.4.1 compile()	29
5.5.4.2 compilePreprocess()	30
5.5.4.3 dump()	30
5.6 Tang::AstNodeBoolean Class Reference	31
5.6.1 Detailed Description	32
5.6.2 Member Enumeration Documentation	32
5.6.2.1 PreprocessState	32
5.6.3 Constructor & Destructor Documentation	32
5.6.3.1 AstNodeBoolean()	32
5.6.4 Member Function Documentation	32
5.6.4.1 compile()	32
5.6.4.2 compilePreprocess()	34
5.6.4.3 dump()	34
5.7 Tang::AstNodeBreak Class Reference	35
5.7.1 Detailed Description	36
5.7.2 Member Enumeration Documentation	36
5.7.2.1 PreprocessState	36
5.7.3 Constructor & Destructor Documentation	36
5.7.3.1 AstNodeBreak()	36
5.7.4 Member Function Documentation	37
5.7.4.1 compile()	37
5.7.4.2 compilePreprocess()	37
5.7.4.3 dump()	38

5.8 Tang::AstNodeCast Class Reference	38
5.8.1 Detailed Description	40
5.8.2 Member Enumeration Documentation	40
5.8.2.1 PreprocessState	40
5.8.2.2 Type	40
5.8.3 Constructor & Destructor Documentation	40
5.8.3.1 AstNodeCast()	40
5.8.4 Member Function Documentation	41
5.8.4.1 compile()	41
5.8.4.2 compilePreprocess()	41
5.8.4.3 dump()	42
5.9 Tang::AstNodeContinue Class Reference	42
5.9.1 Detailed Description	43
5.9.2 Member Enumeration Documentation	43
5.9.2.1 PreprocessState	43
5.9.3 Constructor & Destructor Documentation	44
5.9.3.1 AstNodeContinue()	44
5.9.4 Member Function Documentation	44
5.9.4.1 compile()	44
5.9.4.2 compilePreprocess()	45
5.9.4.3 dump()	45
5.10 Tang::AstNodeDoWhile Class Reference	46
5.10.1 Detailed Description	47
5.10.2 Member Enumeration Documentation	47
5.10.2.1 PreprocessState	47
5.10.3 Constructor & Destructor Documentation	47
5.10.3.1 AstNodeDoWhile()	47
5.10.4 Member Function Documentation	48
5.10.4.1 compile()	48
5.10.4.2 compilePreprocess()	48
5.10.4.3 dump()	49
5.11 Tang::AstNodeFloat Class Reference	49
5.11.1 Detailed Description	50
5.11.2 Member Enumeration Documentation	50
5.11.2.1 PreprocessState	50
5.11.3 Constructor & Destructor Documentation	51
5.11.3.1 AstNodeFloat()	51
5.11.4 Member Function Documentation	51
5.11.4.1 compile()	51
5.11.4.2 compilePreprocess()	52
5.11.4.3 dump()	52
5.12 Tang::AstNodeFor Class Reference	53

5.12.1 Detailed Description	54
5.12.2 Member Enumeration Documentation	54
5.12.2.1 PreprocessState	54
5.12.3 Constructor & Destructor Documentation	54
5.12.3.1 AstNodeFor()	54
5.12.4 Member Function Documentation	55
5.12.4.1 compile()	55
5.12.4.2 compilePreprocess()	56
5.12.4.3 dump()	56
5.13 Tang::AstNodeFunctionCall Class Reference	56
5.13.1 Detailed Description	58
5.13.2 Member Enumeration Documentation	58
5.13.2.1 PreprocessState	58
5.13.3 Constructor & Destructor Documentation	58
5.13.3.1 AstNodeFunctionCall()	58
5.13.4 Member Function Documentation	58
5.13.4.1 compile()	59
5.13.4.2 compilePreprocess()	59
5.13.4.3 dump()	59
5.14 Tang::AstNodeFunctionDeclaration Class Reference	60
5.14.1 Detailed Description	61
5.14.2 Member Enumeration Documentation	61
5.14.2.1 PreprocessState	61
5.14.3 Constructor & Destructor Documentation	61
5.14.3.1 AstNodeFunctionDeclaration()	61
5.14.4 Member Function Documentation	62
5.14.4.1 compile()	62
5.14.4.2 compilePreprocess()	63
5.14.4.3 dump()	63
5.15 Tang::AstNodeIdentifier Class Reference	64
5.15.1 Detailed Description	65
5.15.2 Member Enumeration Documentation	65
5.15.2.1 PreprocessState	65
5.15.3 Constructor & Destructor Documentation	65
5.15.3.1 AstNodeIdentifier()	65
5.15.4 Member Function Documentation	66
5.15.4.1 compile()	66
5.15.4.2 compilePreprocess()	66
5.15.4.3 dump()	67
5.16 Tang::AstNodeIfElse Class Reference	68
5.16.1 Detailed Description	69
5.16.2 Member Enumeration Documentation	69

5.16.2.1 PreprocessState	69
5.16.3 Constructor & Destructor Documentation	69
5.16.3.1 AstNodeIfElse() [1/2]	69
5.16.3.2 AstNodeIfElse() [2/2]	70
5.16.4 Member Function Documentation	70
5.16.4.1 compile()	70
5.16.4.2 compilePreprocess()	71
5.16.4.3 dump()	71
5.17 Tang::AstNodeIndex Class Reference	71
5.17.1 Detailed Description	73
5.17.2 Member Enumeration Documentation	73
5.17.2.1 PreprocessState	73
5.17.3 Constructor & Destructor Documentation	73
5.17.3.1 AstNodeIndex()	73
5.17.4 Member Function Documentation	73
5.17.4.1 compile()	74
5.17.4.2 compilePreprocess()	74
5.17.4.3 dump()	74
5.17.4.4 getCollection()	75
5.17.4.5 getIndex()	75
5.18 Tang::AstNodeInteger Class Reference	76
5.18.1 Detailed Description	77
5.18.2 Member Enumeration Documentation	77
5.18.2.1 PreprocessState	77
5.18.3 Constructor & Destructor Documentation	77
5.18.3.1 AstNodeInteger()	77
5.18.4 Member Function Documentation	77
5.18.4.1 compile()	78
5.18.4.2 compilePreprocess()	78
5.18.4.3 dump()	78
5.19 Tang::AstNodePrint Class Reference	79
5.19.1 Detailed Description	80
5.19.2 Member Enumeration Documentation	80
5.19.2.1 PreprocessState	80
5.19.2.2 Type	80
5.19.3 Constructor & Destructor Documentation	81
5.19.3.1 AstNodePrint()	81
5.19.4 Member Function Documentation	81
5.19.4.1 compile()	81
5.19.4.2 compilePreprocess()	82
5.19.4.3 dump()	82
5.20 Tang::AstNodeReturn Class Reference	83

5.20.1 Detailed Description	84
5.20.2 Member Enumeration Documentation	84
5.20.2.1 PreprocessState	84
5.20.3 Constructor & Destructor Documentation	84
5.20.3.1 AstNodeReturn()	84
5.20.4 Member Function Documentation	84
5.20.4.1 compile()	84
5.20.4.2 compilePreprocess()	86
5.20.4.3 dump()	86
5.21 Tang::AstNodeSlice Class Reference	87
5.21.1 Detailed Description	88
5.21.2 Member Enumeration Documentation	88
5.21.2.1 PreprocessState	88
5.21.3 Constructor & Destructor Documentation	88
5.21.3.1 AstNodeSlice()	88
5.21.4 Member Function Documentation	89
5.21.4.1 compile()	89
5.21.4.2 compilePreprocess()	90
5.21.4.3 dump()	90
5.22 Tang::AstNodeString Class Reference	90
5.22.1 Detailed Description	92
5.22.2 Member Enumeration Documentation	92
5.22.2.1 PreprocessState	92
5.22.3 Constructor & Destructor Documentation	92
5.22.3.1 AstNodeString()	92
5.22.4 Member Function Documentation	92
5.22.4.1 compile()	92
5.22.4.2 compileLiteral()	94
5.22.4.3 compilePreprocess()	95
5.22.4.4 dump()	95
5.23 Tang::AstNodeTernary Class Reference	96
5.23.1 Detailed Description	97
5.23.2 Member Enumeration Documentation	97
5.23.2.1 PreprocessState	97
5.23.3 Constructor & Destructor Documentation	97
5.23.3.1 AstNodeTernary()	97
5.23.4 Member Function Documentation	97
5.23.4.1 compile()	98
5.23.4.2 compilePreprocess()	98
5.23.4.3 dump()	98
5.24 Tang::AstNodeUnary Class Reference	99
5.24.1 Detailed Description	100

5.24.2 Member Enumeration Documentation	100
5.24.2.1 Operator	100
5.24.2.2 PreprocessState	100
5.24.3 Constructor & Destructor Documentation	101
5.24.3.1 AstNodeUnary()	101
5.24.4 Member Function Documentation	101
5.24.4.1 compile()	101
5.24.4.2 compilePreprocess()	102
5.24.4.3 dump()	102
5.25 Tang::AstNodeWhile Class Reference	103
5.25.1 Detailed Description	104
5.25.2 Member Enumeration Documentation	104
5.25.2.1 PreprocessState	104
5.25.3 Constructor & Destructor Documentation	104
5.25.3.1 AstNodeWhile()	104
5.25.4 Member Function Documentation	104
5.25.4.1 compile()	105
5.25.4.2 compilePreprocess()	105
5.25.4.3 dump()	106
5.26 Tang::ComputedExpression Class Reference	106
5.26.1 Detailed Description	108
5.26.2 Member Function Documentation	108
5.26.2.1 __add()	108
5.26.2.2 __asCode()	109
5.26.2.3 __assign_index()	109
5.26.2.4 __boolean()	110
5.26.2.5 __divide()	110
5.26.2.6 __equal()	110
5.26.2.7 __float()	111
5.26.2.8 __index()	111
5.26.2.9 __integer()	111
5.26.2.10 __lessThan()	112
5.26.2.11 __modulo()	112
5.26.2.12 __multiply()	112
5.26.2.13 __negative()	113
5.26.2.14 __not()	113
5.26.2.15 __slice()	113
5.26.2.16 __string()	114
5.26.2.17 __subtract()	114
5.26.2.18 dump()	115
5.26.2.19 is_equal() [1/6]	115
5.26.2.20 is_equal() [2/6]	115

5.26.2.21 <code>is_equal()</code> [3/6]	116
5.26.2.22 <code>is_equal()</code> [4/6]	116
5.26.2.23 <code>is_equal()</code> [5/6]	116
5.26.2.24 <code>is_equal()</code> [6/6]	117
5.26.2.25 <code>isCopyNeeded()</code>	117
5.26.2.26 <code>makeCopy()</code>	118
5.27 Tang::ComputedExpressionArray Class Reference	118
5.27.1 Detailed Description	120
5.27.2 Constructor & Destructor Documentation	120
5.27.2.1 <code>ComputedExpressionArray()</code>	120
5.27.3 Member Function Documentation	120
5.27.3.1 <code>__add()</code>	120
5.27.3.2 <code>__asCode()</code>	121
5.27.3.3 <code>__assign_index()</code>	121
5.27.3.4 <code>__boolean()</code>	121
5.27.3.5 <code>__divide()</code>	122
5.27.3.6 <code>__equal()</code>	122
5.27.3.7 <code>__float()</code>	123
5.27.3.8 <code>__index()</code>	123
5.27.3.9 <code>__integer()</code>	123
5.27.3.10 <code>__lessThan()</code>	123
5.27.3.11 <code>__modulo()</code>	124
5.27.3.12 <code>__multiply()</code>	124
5.27.3.13 <code>__negative()</code>	125
5.27.3.14 <code>__not()</code>	125
5.27.3.15 <code>__slice()</code>	125
5.27.3.16 <code>__string()</code>	126
5.27.3.17 <code>__subtract()</code>	126
5.27.3.18 <code>dump()</code>	127
5.27.3.19 <code>is_equal()</code> [1/6]	127
5.27.3.20 <code>is_equal()</code> [2/6]	127
5.27.3.21 <code>is_equal()</code> [3/6]	128
5.27.3.22 <code>is_equal()</code> [4/6]	128
5.27.3.23 <code>is_equal()</code> [5/6]	128
5.27.3.24 <code>is_equal()</code> [6/6]	129
5.27.3.25 <code>isCopyNeeded()</code>	129
5.27.3.26 <code>makeCopy()</code>	130
5.28 Tang::ComputedExpressionBoolean Class Reference	130
5.28.1 Detailed Description	132
5.28.2 Constructor & Destructor Documentation	132
5.28.2.1 <code>ComputedExpressionBoolean()</code>	132
5.28.3 Member Function Documentation	132

5.28.3.1	__add()	132
5.28.3.2	__asCode()	133
5.28.3.3	__assign_index()	133
5.28.3.4	__boolean()	133
5.28.3.5	__divide()	134
5.28.3.6	__equal()	134
5.28.3.7	__float()	134
5.28.3.8	__index()	135
5.28.3.9	__integer()	135
5.28.3.10	__lessThan()	135
5.28.3.11	__modulo()	136
5.28.3.12	__multiply()	136
5.28.3.13	__negative()	136
5.28.3.14	__not()	137
5.28.3.15	__slice()	137
5.28.3.16	__string()	137
5.28.3.17	__subtract()	138
5.28.3.18	dump()	138
5.28.3.19	is_equal() [1/6]	138
5.28.3.20	is_equal() [2/6]	139
5.28.3.21	is_equal() [3/6]	139
5.28.3.22	is_equal() [4/6]	139
5.28.3.23	is_equal() [5/6]	140
5.28.3.24	is_equal() [6/6]	140
5.28.3.25	isCopyNeeded()	140
5.28.3.26	makeCopy()	141
5.29	Tang::ComputedExpressionCompiledFunction Class Reference	141
5.29.1	Detailed Description	143
5.29.2	Constructor & Destructor Documentation	143
5.29.2.1	ComputedExpressionCompiledFunction()	143
5.29.3	Member Function Documentation	144
5.29.3.1	__add()	144
5.29.3.2	__asCode()	144
5.29.3.3	__assign_index()	144
5.29.3.4	__boolean()	145
5.29.3.5	__divide()	145
5.29.3.6	__equal()	145
5.29.3.7	__float()	146
5.29.3.8	__index()	146
5.29.3.9	__integer()	147
5.29.3.10	__lessThan()	147
5.29.3.11	__modulo()	147

5.29.3.12	__multiply()	148
5.29.3.13	__negative()	148
5.29.3.14	__not()	148
5.29.3.15	__slice()	149
5.29.3.16	__string()	149
5.29.3.17	__subtract()	149
5.29.3.18	dump()	150
5.29.3.19	is_equal() [1/6]	150
5.29.3.20	is_equal() [2/6]	151
5.29.3.21	is_equal() [3/6]	152
5.29.3.22	is_equal() [4/6]	152
5.29.3.23	is_equal() [5/6]	153
5.29.3.24	is_equal() [6/6]	153
5.29.3.25	isCopyNeeded()	153
5.29.3.26	makeCopy()	154
5.30	Tang::ComputedExpressionError Class Reference	154
5.30.1	Detailed Description	156
5.30.2	Constructor & Destructor Documentation	156
5.30.2.1	ComputedExpressionError()	156
5.30.3	Member Function Documentation	156
5.30.3.1	__add()	156
5.30.3.2	__asCode()	157
5.30.3.3	__assign_index()	157
5.30.3.4	__boolean()	157
5.30.3.5	__divide()	158
5.30.3.6	__equal()	158
5.30.3.7	__float()	158
5.30.3.8	__index()	159
5.30.3.9	__integer()	159
5.30.3.10	__lessThan()	159
5.30.3.11	__modulo()	160
5.30.3.12	__multiply()	160
5.30.3.13	__negative()	160
5.30.3.14	__not()	161
5.30.3.15	__slice()	161
5.30.3.16	__string()	161
5.30.3.17	__subtract()	162
5.30.3.18	dump()	162
5.30.3.19	is_equal() [1/6]	162
5.30.3.20	is_equal() [2/6]	163
5.30.3.21	is_equal() [3/6]	163
5.30.3.22	is_equal() [4/6]	163

5.30.3.23 is_equal() [5 / 6]	164
5.30.3.24 is_equal() [6 / 6]	164
5.30.3.25 isCopyNeeded()	164
5.30.3.26 makeCopy()	165
5.31 Tang::ComputedExpressionFloat Class Reference	165
5.31.1 Detailed Description	167
5.31.2 Constructor & Destructor Documentation	167
5.31.2.1 ComputedExpressionFloat()	167
5.31.3 Member Function Documentation	168
5.31.3.1 __add()	168
5.31.3.2 __asCode()	168
5.31.3.3 __assign_index()	168
5.31.3.4 __boolean()	169
5.31.3.5 __divide()	169
5.31.3.6 __equal()	169
5.31.3.7 __float()	170
5.31.3.8 __index()	170
5.31.3.9 __integer()	170
5.31.3.10 __lessThan()	171
5.31.3.11 __modulo()	171
5.31.3.12 __multiply()	171
5.31.3.13 __negative()	172
5.31.3.14 __not()	172
5.31.3.15 __slice()	172
5.31.3.16 __string()	173
5.31.3.17 __subtract()	173
5.31.3.18 dump()	174
5.31.3.19 is_equal() [1 / 6]	174
5.31.3.20 is_equal() [2 / 6]	174
5.31.3.21 is_equal() [3 / 6]	175
5.31.3.22 is_equal() [4 / 6]	175
5.31.3.23 is_equal() [5 / 6]	176
5.31.3.24 is_equal() [6 / 6]	176
5.31.3.25 isCopyNeeded()	176
5.31.3.26 makeCopy()	177
5.32 Tang::ComputedExpressionInteger Class Reference	177
5.32.1 Detailed Description	179
5.32.2 Constructor & Destructor Documentation	179
5.32.2.1 ComputedExpressionInteger()	179
5.32.3 Member Function Documentation	179
5.32.3.1 __add()	179
5.32.3.2 __asCode()	180

5.32.3.3 <code>__assign_index()</code>	180
5.32.3.4 <code>__boolean()</code>	181
5.32.3.5 <code>__divide()</code>	181
5.32.3.6 <code>__equal()</code>	181
5.32.3.7 <code>__float()</code>	182
5.32.3.8 <code>__index()</code>	182
5.32.3.9 <code>__integer()</code>	182
5.32.3.10 <code>__lessThan()</code>	182
5.32.3.11 <code>__modulo()</code>	183
5.32.3.12 <code>__multiply()</code>	183
5.32.3.13 <code>__negative()</code>	184
5.32.3.14 <code>__not()</code>	184
5.32.3.15 <code>__slice()</code>	184
5.32.3.16 <code>__string()</code>	185
5.32.3.17 <code>__subtract()</code>	185
5.32.3.18 <code>dump()</code>	186
5.32.3.19 <code>is_equal()</code> [1/6]	186
5.32.3.20 <code>is_equal()</code> [2/6]	186
5.32.3.21 <code>is_equal()</code> [3/6]	187
5.32.3.22 <code>is_equal()</code> [4/6]	187
5.32.3.23 <code>is_equal()</code> [5/6]	187
5.32.3.24 <code>is_equal()</code> [6/6]	188
5.32.3.25 <code>isCopyNeeded()</code>	188
5.32.3.26 <code>makeCopy()</code>	188
5.33 Tang::ComputedExpressionString Class Reference	189
5.33.1 Detailed Description	190
5.33.2 Constructor & Destructor Documentation	191
5.33.2.1 <code>ComputedExpressionString()</code>	191
5.33.3 Member Function Documentation	191
5.33.3.1 <code>__add()</code>	191
5.33.3.2 <code>__asCode()</code>	191
5.33.3.3 <code>__assign_index()</code>	192
5.33.3.4 <code>__boolean()</code>	192
5.33.3.5 <code>__divide()</code>	193
5.33.3.6 <code>__equal()</code>	193
5.33.3.7 <code>__float()</code>	194
5.33.3.8 <code>__index()</code>	194
5.33.3.9 <code>__integer()</code>	195
5.33.3.10 <code>__lessThan()</code>	195
5.33.3.11 <code>__modulo()</code>	195
5.33.3.12 <code>__multiply()</code>	196
5.33.3.13 <code>__negative()</code>	196

5.33.3.14	<code>__not()</code>	196
5.33.3.15	<code>__slice()</code>	197
5.33.3.16	<code>__string()</code>	198
5.33.3.17	<code>__subtract()</code>	198
5.33.3.18	<code>dump()</code>	198
5.33.3.19	<code>is_equal()</code> [1/6]	198
5.33.3.20	<code>is_equal()</code> [2/6]	199
5.33.3.21	<code>is_equal()</code> [3/6]	199
5.33.3.22	<code>is_equal()</code> [4/6]	200
5.33.3.23	<code>is_equal()</code> [5/6]	200
5.33.3.24	<code>is_equal()</code> [6/6]	200
5.33.3.25	<code>isCopyNeeded()</code>	201
5.33.3.26	<code>makeCopy()</code>	201
5.34	Tang::Error Class Reference	202
5.34.1	Detailed Description	203
5.34.2	Constructor & Destructor Documentation	203
5.34.2.1	<code>Error()</code> [1/2]	203
5.34.2.2	<code>Error()</code> [2/2]	203
5.34.3	Friends And Related Function Documentation	203
5.34.3.1	<code>operator<<</code>	204
5.35	Tang::GarbageCollected Class Reference	204
5.35.1	Detailed Description	206
5.35.2	Constructor & Destructor Documentation	206
5.35.2.1	<code>GarbageCollected()</code> [1/3]	206
5.35.2.2	<code>GarbageCollected()</code> [2/3]	207
5.35.2.3	<code>~GarbageCollected()</code>	207
5.35.2.4	<code>GarbageCollected()</code> [3/3]	207
5.35.3	Member Function Documentation	207
5.35.3.1	<code>isCopyNeeded()</code>	207
5.35.3.2	<code>make()</code>	208
5.35.3.3	<code>makeCopy()</code>	208
5.35.3.4	<code>operator"!()</code>	209
5.35.3.5	<code>operator"!=(</code>	209
5.35.3.6	<code>operator%(</code>	210
5.35.3.7	<code>operator*(</code> [1/2]	211
5.35.3.8	<code>operator*(</code> [2/2]	211
5.35.3.9	<code>operator+(</code>	211
5.35.3.10	<code>operator-(</code> [1/2]	212
5.35.3.11	<code>operator-(</code> [2/2]	212
5.35.3.12	<code>operator->()</code>	213
5.35.3.13	<code>operator/(</code>	213
5.35.3.14	<code>operator<()</code>	214

5.35.3.15 operator<=()	214
5.35.3.16 operator=() [1/2]	215
5.35.3.17 operator=() [2/2]	215
5.35.3.18 operator==() [1/8]	215
5.35.3.19 operator==() [2/8]	216
5.35.3.20 operator==() [3/8]	216
5.35.3.21 operator==() [4/8]	216
5.35.3.22 operator==() [5/8]	217
5.35.3.23 operator==() [6/8]	217
5.35.3.24 operator==() [7/8]	218
5.35.3.25 operator==() [8/8]	218
5.35.3.26 operator>()	219
5.35.3.27 operator>=()	219
5.35.4 Friends And Related Function Documentation	219
5.35.4.1 operator<<	220
5.36 Tang::HtmlEscape Class Reference	220
5.36.1 Detailed Description	221
5.36.2 Constructor & Destructor Documentation	221
5.36.2.1 HtmlEscape()	221
5.36.3 Member Function Documentation	222
5.36.3.1 get_next_token()	222
5.37 Tang::location Class Reference	222
5.37.1 Detailed Description	223
5.38 Tang::position Class Reference	224
5.38.1 Detailed Description	225
5.39 Tang::Program Class Reference	225
5.39.1 Detailed Description	227
5.39.2 Member Enumeration Documentation	227
5.39.2.1 CodeType	227
5.39.3 Constructor & Destructor Documentation	227
5.39.3.1 Program()	227
5.39.4 Member Function Documentation	228
5.39.4.1 addBreak()	228
5.39.4.2 addBytecode()	228
5.39.4.3 addContinue()	228
5.39.4.4 addIdentifier()	229
5.39.4.5 addIdentifierAssigned()	229
5.39.4.6 addString()	229
5.39.4.7 dumpBytecode()	229
5.39.4.8 execute()	230
5.39.4.9 getAst()	230
5.39.4.10 getBytecode()	230

5.39.4.11 getCode()	231
5.39.4.12 getIdentifiers()	231
5.39.4.13 getIdentifiersAssigned()	231
5.39.4.14 getResult()	231
5.39.4.15 getStrings()	232
5.39.4.16 popBreakStack()	232
5.39.4.17 popContinueStack()	232
5.39.4.18 pushEnvironment()	233
5.39.4.19 setFunctionStackDeclaration()	233
5.39.4.20 setJumpTarget()	234
5.39.5 Member Data Documentation	234
5.39.5.1 functionsDeclared	234
5.40 Tang::SingletonObjectPool< T > Class Template Reference	234
5.40.1 Detailed Description	235
5.40.2 Member Function Documentation	235
5.40.2.1 get()	235
5.40.2.2 getInstance()	235
5.40.2.3 recycle()	235
5.41 Tang::TangBase Class Reference	236
5.41.1 Detailed Description	236
5.41.2 Constructor & Destructor Documentation	236
5.41.2.1 TangBase()	236
5.41.3 Member Function Documentation	236
5.41.3.1 compileScript()	236
5.42 Tang::TangScanner Class Reference	237
5.42.1 Detailed Description	238
5.42.2 Constructor & Destructor Documentation	238
5.42.2.1 TangScanner()	238
5.42.3 Member Function Documentation	238
5.42.3.1 get_next_token()	238
5.43 Tang::Unescape Class Reference	239
5.43.1 Detailed Description	240
5.43.2 Constructor & Destructor Documentation	240
5.43.2.1 Unescape()	240
5.43.3 Member Function Documentation	240
5.43.3.1 get_next_token()	240
5.44 Tang::UnicodeString Class Reference	241
5.44.1 Detailed Description	241
5.44.2 Constructor & Destructor Documentation	241
5.44.2.1 UnicodeString()	241
5.44.3 Member Function Documentation	242
5.44.3.1 bytesLength()	242

5.44.3.2 length()	242
5.44.3.3 operator std::string()	242
5.44.3.4 operator+()	242
5.44.3.5 operator<()	243
5.44.3.6 operator==()	243
5.44.3.7 substr()	243
6 File Documentation	245
6.1 build/generated/location.hh File Reference	245
6.1.1 Detailed Description	246
6.1.2 Function Documentation	246
6.1.2.1 operator<<() [1/2]	246
6.1.2.2 operator<<() [2/2]	247
6.2 include/astNode.hpp File Reference	247
6.2.1 Detailed Description	248
6.3 include/astNodeArray.hpp File Reference	248
6.3.1 Detailed Description	249
6.4 include/astNodeAssign.hpp File Reference	249
6.4.1 Detailed Description	250
6.5 include/astNodeBinary.hpp File Reference	250
6.5.1 Detailed Description	251
6.6 include/astNodeBlock.hpp File Reference	251
6.6.1 Detailed Description	252
6.7 include/astNodeBoolean.hpp File Reference	252
6.7.1 Detailed Description	253
6.8 include/astNodeBreak.hpp File Reference	253
6.8.1 Detailed Description	254
6.9 include/astNodeCast.hpp File Reference	254
6.9.1 Detailed Description	255
6.10 include/astNodeContinue.hpp File Reference	255
6.10.1 Detailed Description	256
6.11 include/astNodeDoWhile.hpp File Reference	256
6.11.1 Detailed Description	257
6.12 include/astNodeFloat.hpp File Reference	257
6.12.1 Detailed Description	258
6.13 include/astNodeFor.hpp File Reference	258
6.13.1 Detailed Description	259
6.14 include/astNodeFunctionCall.hpp File Reference	259
6.14.1 Detailed Description	260
6.15 include/astNodeFunctionDeclaration.hpp File Reference	260
6.15.1 Detailed Description	261
6.16 include/astNodeIdentifier.hpp File Reference	261

6.16.1 Detailed Description	262
6.17 include/astNodeIfElse.hpp File Reference	262
6.17.1 Detailed Description	263
6.18 include/astNodeIndex.hpp File Reference	263
6.18.1 Detailed Description	264
6.19 include/astNodeInteger.hpp File Reference	264
6.19.1 Detailed Description	265
6.20 include/astNodePrint.hpp File Reference	265
6.20.1 Detailed Description	266
6.21 include/astNodeReturn.hpp File Reference	266
6.21.1 Detailed Description	267
6.22 include/astNodeSlice.hpp File Reference	267
6.22.1 Detailed Description	268
6.23 include/astNodeString.hpp File Reference	268
6.23.1 Detailed Description	269
6.24 include/astNodeTernary.hpp File Reference	269
6.24.1 Detailed Description	270
6.25 include/astNodeUnary.hpp File Reference	270
6.25.1 Detailed Description	271
6.26 include/astNodeWhile.hpp File Reference	271
6.26.1 Detailed Description	272
6.27 include/computedExpression.hpp File Reference	272
6.27.1 Detailed Description	272
6.28 include/computedExpressionArray.hpp File Reference	273
6.28.1 Detailed Description	273
6.29 include/computedExpressionBoolean.hpp File Reference	274
6.29.1 Detailed Description	274
6.30 include/computedExpressionCompiledFunction.hpp File Reference	275
6.30.1 Detailed Description	275
6.31 include/computedExpressionError.hpp File Reference	276
6.31.1 Detailed Description	276
6.32 include/computedExpressionFloat.hpp File Reference	277
6.32.1 Detailed Description	277
6.33 include/computedExpressionInteger.hpp File Reference	278
6.33.1 Detailed Description	278
6.34 include/computedExpressionString.hpp File Reference	279
6.34.1 Detailed Description	279
6.35 include/error.hpp File Reference	279
6.35.1 Detailed Description	280
6.36 include/garbageCollected.hpp File Reference	280
6.36.1 Detailed Description	281
6.37 include/htmlEscape.hpp File Reference	281

6.37.1 Detailed Description	282
6.38 include/macros.hpp File Reference	283
6.38.1 Detailed Description	283
6.39 include/opcode.hpp File Reference	283
6.39.1 Detailed Description	284
6.39.2 Enumeration Type Documentation	284
6.39.2.1 Opcode	284
6.40 include/program.hpp File Reference	285
6.40.1 Detailed Description	286
6.41 include/singletonObjectPool.hpp File Reference	286
6.41.1 Detailed Description	287
6.42 include/tang.hpp File Reference	287
6.42.1 Detailed Description	287
6.43 include/tangBase.hpp File Reference	288
6.43.1 Detailed Description	288
6.44 include/tangScanner.hpp File Reference	289
6.44.1 Detailed Description	290
6.45 include/unescape.hpp File Reference	290
6.45.1 Detailed Description	291
6.46 include/unicodeString.hpp File Reference	291
6.46.1 Detailed Description	292
6.46.2 Function Documentation	292
6.46.2.1 htmlEscape()	292
6.46.2.2 unescape()	292
6.47 src/astNode.cpp File Reference	293
6.47.1 Detailed Description	293
6.48 src/astNodeArray.cpp File Reference	294
6.48.1 Detailed Description	294
6.49 src/astNodeAssign.cpp File Reference	294
6.49.1 Detailed Description	295
6.50 src/astNodeBinary.cpp File Reference	295
6.50.1 Detailed Description	296
6.51 src/astNodeBlock.cpp File Reference	296
6.51.1 Detailed Description	296
6.52 src/astNodeBoolean.cpp File Reference	296
6.52.1 Detailed Description	297
6.53 src/astNodeBreak.cpp File Reference	297
6.53.1 Detailed Description	298
6.54 src/astNodeCast.cpp File Reference	298
6.54.1 Detailed Description	298
6.55 src/astNodeContinue.cpp File Reference	298
6.55.1 Detailed Description	299

6.56 src/astNodeDoWhile.cpp File Reference	299
6.56.1 Detailed Description	300
6.57 src/astNodeFloat.cpp File Reference	300
6.57.1 Detailed Description	301
6.58 src/astNodeFor.cpp File Reference	301
6.58.1 Detailed Description	301
6.59 src/astNodeFunctionCall.cpp File Reference	301
6.59.1 Detailed Description	302
6.60 src/astNodeFunctionDeclaration.cpp File Reference	302
6.60.1 Detailed Description	303
6.61 src/astNodeIdentifier.cpp File Reference	303
6.61.1 Detailed Description	304
6.62 src/astNodeIfElse.cpp File Reference	304
6.62.1 Detailed Description	304
6.63 src/astNodeIndex.cpp File Reference	304
6.63.1 Detailed Description	305
6.64 src/astNodeInteger.cpp File Reference	305
6.64.1 Detailed Description	306
6.65 src/astNodePrint.cpp File Reference	306
6.65.1 Detailed Description	306
6.66 src/astNodeReturn.cpp File Reference	306
6.66.1 Detailed Description	307
6.67 src/astNodeSlice.cpp File Reference	307
6.67.1 Detailed Description	308
6.68 src/astNodeString.cpp File Reference	308
6.68.1 Detailed Description	309
6.69 src/astNodeTernary.cpp File Reference	309
6.69.1 Detailed Description	310
6.70 src/astNodeUnary.cpp File Reference	310
6.70.1 Detailed Description	310
6.71 src/astNodeWhile.cpp File Reference	310
6.71.1 Detailed Description	311
6.72 src/computedExpression.cpp File Reference	311
6.72.1 Detailed Description	312
6.73 src/computedExpressionArray.cpp File Reference	312
6.73.1 Detailed Description	313
6.74 src/computedExpressionBoolean.cpp File Reference	313
6.74.1 Detailed Description	313
6.75 src/computedExpressionCompiledFunction.cpp File Reference	313
6.75.1 Detailed Description	314
6.76 src/computedExpressionError.cpp File Reference	314
6.76.1 Detailed Description	315

6.77 src/computedExpressionFloat.cpp File Reference	315
6.77.1 Detailed Description	315
6.78 src/computedExpressionInteger.cpp File Reference	315
6.78.1 Detailed Description	316
6.79 src/computedExpressionString.cpp File Reference	316
6.79.1 Detailed Description	316
6.80 src/error.cpp File Reference	317
6.80.1 Detailed Description	317
6.80.2 Function Documentation	317
6.80.2.1 operator<<()	317
6.81 src/program-dumpBytecode.cpp File Reference	318
6.81.1 Detailed Description	318
6.81.2 Macro Definition Documentation	318
6.81.2.1 DUMPPROGRAMCHECK	319
6.82 src/program-execute.cpp File Reference	319
6.82.1 Detailed Description	320
6.82.2 Macro Definition Documentation	320
6.82.2.1 EXECUTEPROGRAMCHECK	320
6.82.2.2 STACKCHECK	320
6.83 src/program.cpp File Reference	320
6.83.1 Detailed Description	321
6.84 src/tangBase.cpp File Reference	321
6.84.1 Detailed Description	322
6.85 src/unicodeString.cpp File Reference	322
6.85.1 Detailed Description	322
6.86 test/test.cpp File Reference	322
6.86.1 Detailed Description	324
6.87 test/testGarbageCollected.cpp File Reference	324
6.87.1 Detailed Description	325
6.88 test/testSingletonObjectPool.cpp File Reference	325
6.88.1 Detailed Description	325
6.89 test/testUnicodeString.cpp File Reference	326
6.89.1 Detailed Description	326
Index	327

Chapter 1

Tang: A Template Language

1.1 Quick Description

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

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

1.2 Features

The following features are planned:

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

1.3 License

MIT License

Copyright (c) 2022 Corey Pennycuff

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

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

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Tang::AstNode	11
Tang::AstNodeArray	16
Tang::AstNodeAssign	20
Tang::AstNodeBinary	23
Tang::AstNodeBlock	27
Tang::AstNodeBoolean	31
Tang::AstNodeBreak	35
Tang::AstNodeCast	38
Tang::AstNodeContinue	42
Tang::AstNodeDoWhile	46
Tang::AstNodeFloat	49
Tang::AstNodeFor	53
Tang::AstNodeFunctionCall	56
Tang::AstNodeFunctionDeclaration	60
Tang::AstNodeIdentifier	64
Tang::AstNodeIfElse	68
Tang::AstNodeIndex	71
Tang::AstNodeInteger	76
Tang::AstNodePrint	79
Tang::AstNodeReturn	83
Tang::AstNodeSlice	87
Tang::AstNodeString	90
Tang::AstNodeTernary	96
Tang::AstNodeUnary	99
Tang::AstNodeWhile	103
Tang::ComputedExpression	106
Tang::ComputedExpressionArray	118
Tang::ComputedExpressionBoolean	130
Tang::ComputedExpressionCompiledFunction	141
Tang::ComputedExpressionError	154
Tang::ComputedExpressionFloat	165
Tang::ComputedExpressionInteger	177
Tang::ComputedExpressionString	189
Tang::Error	202
Tang::GarbageCollected	204

Tang::location	222
Tang::position	224
Tang::Program	225
Tang::SingletonObjectPool< T >	234
Tang::TangBase	236
TangHtmlEscapeFlexLexer	
Tang::HtmlEscape	220
TangTangFlexLexer	
Tang::TangScanner	237
TangUnescapeFlexLexer	
Tang::Unescape	239
Tang::UnicodeString	241

Chapter 3

Class Index

3.1 Class List

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

Tang::AstNode	Base class for representing nodes of an Abstract Syntax Tree (AST)	11
Tang::AstNodeArray	An AstNode that represents an array literal	16
Tang::AstNodeAssign	An AstNode that represents a binary expression	20
Tang::AstNodeBinary	An AstNode that represents a binary expression	23
Tang::AstNodeBlock	An AstNode that represents a code block	27
Tang::AstNodeBoolean	An AstNode that represents a boolean literal	31
Tang::AstNodeBreak	An AstNode that represents a <code>break</code> statement	35
Tang::AstNodeCast	An AstNode that represents a typecast of an expression	38
Tang::AstNodeContinue	An AstNode that represents a <code>continue</code> statement	42
Tang::AstNodeDoWhile	An AstNode that represents a <code>do..while</code> statement	46
Tang::AstNodeFloat	An AstNode that represents an float literal	49
Tang::AstNodeFor	An AstNode that represents an <code>if()</code> statement	53
Tang::AstNodeFunctionCall	An AstNode that represents a function call	56
Tang::AstNodeFunctionDeclaration	An AstNode that represents a function declaration	60
Tang::AstNodeIdentifier	An AstNode that represents an identifier	64
Tang::AstNodeIfElse	An AstNode that represents an <code>if..else</code> statement	68
Tang::AstNodeIndex	An AstNode that represents an index into a collection	71
Tang::AstNodeInteger	An AstNode that represents an integer literal	76

Tang::AstNodePrint	
An AstNode that represents a print typeoperation	79
Tang::AstNodeReturn	
An AstNode that represents a <code>return</code> statement	83
Tang::AstNodeSlice	
An AstNode that represents a ternary expression	87
Tang::AstNodeString	
An AstNode that represents a string literal	90
Tang::AstNodeTernary	
An AstNode that represents a ternary expression	96
Tang::AstNodeUnary	
An AstNode that represents a unary negation	99
Tang::AstNodeWhile	
An AstNode that represents a while statement	103
Tang::ComputedExpression	
Represents the result of a computation that has been executed	106
Tang::ComputedExpressionArray	
Represents an Array that is the result of a computation	118
Tang::ComputedExpressionBoolean	
Represents an Boolean that is the result of a computation	130
Tang::ComputedExpressionCompiledFunction	
Represents a Compiled Function declared in the script	141
Tang::ComputedExpressionError	
Represents a Runtime Error	154
Tang::ComputedExpressionFloat	
Represents a Float that is the result of a computation	165
Tang::ComputedExpressionInteger	
Represents an Integer that is the result of a computation	177
Tang::ComputedExpressionString	
Represents a String that is the result of a computation	189
Tang::Error	
Used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error	202
Tang::GarbageCollected	
A container that acts as a resource-counting garbage collector for the specified type	204
Tang::HtmlEscape	
The Flex lexer class for the main Tang language	220
Tang::location	
Two points in a source file	222
Tang::position	
A point in a source file	224
Tang::Program	
Represents a compiled script or template that may be executed	225
Tang::SingletonObjectPool< T >	
A thread-safe, singleton object pool of the designated type	234
Tang::TangBase	
The base class for the Tang programming language	236
Tang::TangScanner	
The Flex lexer class for the main Tang language	237
Tang::Unescape	
The Flex lexer class for the main Tang language	239
Tang::UnicodeString	
Represents a UTF-8 encoded string that is Unicode-aware	241

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	245
include/astNode.hpp	
Declare the <code>Tang::AstNode</code> base class	247
include/astNodeArray.hpp	
Declare the <code>Tang::AstNodeArray</code> class	248
include/astNodeAssign.hpp	
Declare the <code>Tang::AstNodeAssign</code> class	249
include/astNodeBinary.hpp	
Declare the <code>Tang::AstNodeBinary</code> class	250
include/astNodeBlock.hpp	
Declare the <code>Tang::AstNodeBlock</code> class	251
include/astNodeBoolean.hpp	
Declare the <code>Tang::AstNodeBoolean</code> class	252
include/astNodeBreak.hpp	
Declare the <code>Tang::AstNodeBreak</code> class	253
include/astNodeCast.hpp	
Declare the <code>Tang::AstNodeCast</code> class	254
include/astNodeContinue.hpp	
Declare the <code>Tang::AstNodeContinue</code> class	255
include/astNodeDoWhile.hpp	
Declare the <code>Tang::AstNodeDoWhile</code> class	256
include/astNodeFloat.hpp	
Declare the <code>Tang::AstNodeFloat</code> class	257
include/astNodeFor.hpp	
Declare the <code>Tang::AstNodeFor</code> class	258
include/astNodeFunctionCall.hpp	
Declare the <code>Tang::AstNodeFunctionCall</code> class	259
include/astNodeFunctionDeclaration.hpp	
Declare the <code>Tang::AstNodeFunctionDeclaration</code> class	260
include/astNodeIdentifier.hpp	
Declare the <code>Tang::AstNodeIdentifier</code> class	261
include/astNodeIfElse.hpp	
Declare the <code>Tang::AstNodeIfElse</code> class	262
include/astNodeIndex.hpp	
Declare the <code>Tang::AstNodeIndex</code> class	263

include/ astNodeInteger.hpp	Declare the Tang::AstNodeInteger class	264
include/ astNodePrint.hpp	Declare the Tang::AstNodePrint class	265
include/ astNodeReturn.hpp	Declare the Tang::AstNodeReturn class	266
include/ astNodeSlice.hpp	Declare the Tang::AstNodeSlice class	267
include/ astNodeString.hpp	Declare the Tang::AstNodeString class	268
include/ astNodeTernary.hpp	Declare the Tang::AstNodeTernary class	269
include/ astNodeUnary.hpp	Declare the Tang::AstNodeUnary class	270
include/ astNodeWhile.hpp	Declare the Tang::AstNodeWhile class	271
include/ computedExpression.hpp	Declare the Tang::ComputedExpression base class	272
include/ computedExpressionArray.hpp	Declare the Tang::ComputedExpressionArray class	273
include/ computedExpressionBoolean.hpp	Declare the Tang::ComputedExpressionBoolean class	274
include/ computedExpressionCompiledFunction.hpp	Declare the Tang::ComputedExpressionCompiledFunction class	275
include/ computedExpressionError.hpp	Declare the Tang::ComputedExpressionError class	276
include/ computedExpressionFloat.hpp	Declare the Tang::ComputedExpressionFloat class	277
include/ computedExpressionInteger.hpp	Declare the Tang::ComputedExpressionInteger class	278
include/ computedExpressionString.hpp	Declare the Tang::ComputedExpressionString class	279
include/ error.hpp	Declare the Tang::Error class used to describe syntax and runtime errors	279
include/ garbageCollected.hpp	Declare the Tang::GarbageCollected class	280
include/ htmlEscape.hpp	Declare the Tang::HtmlEscape used to tokenize a Tang script	281
include/ macros.hpp	Contains generic macros	283
include/ opcode.hpp	Declare the Opcodes used in the Bytecode representation of a program	283
include/ program.hpp	Declare the Tang::Program class used to compile and execute source code	285
include/ singletonObjectPool.hpp	Declare the Tang::SingletonObjectPool class	286
include/ tang.hpp	Header file supplied for use by 3rd party code so that they can easily include all necessary headers	287
include/ tangBase.hpp	Declare the Tang::TangBase class used to interact with Tang	288
include/ tangScanner.hpp	Declare the Tang::TangScanner used to tokenize a Tang script	289
include/ unescape.hpp	Declare the Tang::Unescape used to tokenize a Tang script	290
include/ unicodeString.hpp	Contains the code to interface with the ICU library	291

src/ astNode.cpp	
Define the Tang::AstNode class	293
src/ astNodeArray.cpp	
Define the Tang::AstNodeArray class	294
src/ astNodeAssign.cpp	
Define the Tang::AstNodeAssign class	294
src/ astNodeBinary.cpp	
Define the Tang::AstNodeBinary class	295
src/ astNodeBlock.cpp	
Define the Tang::AstNodeBlock class	296
src/ astNodeBoolean.cpp	
Define the Tang::AstNodeBoolean class	296
src/ astNodeBreak.cpp	
Define the Tang::AstNodeBreak class	297
src/ astNodeCast.cpp	
Define the Tang::AstNodeCast class	298
src/ astNodeContinue.cpp	
Define the Tang::AstNodeContinue class	298
src/ astNodeDoWhile.cpp	
Define the Tang::AstNodeDoWhile class	299
src/ astNodeFloat.cpp	
Define the Tang::AstNodeFloat class	300
src/ astNodeFor.cpp	
Define the Tang::AstNodeFor class	301
src/ astNodeFunctionCall.cpp	
Define the Tang::AstNodeFunctionCall class	301
src/ astNodeFunctionDeclaration.cpp	
Define the Tang::AstNodeFunctionDeclaration class	302
src/ astNodeIdentifier.cpp	
Define the Tang::AstNodeIdentifier class	303
src/ astNodeIfElse.cpp	
Define the Tang::AstNodeIfElse class	304
src/ astNodeIndex.cpp	
Define the Tang::AstNodeIndex class	304
src/ astNodeInteger.cpp	
Define the Tang::AstNodeInteger class	305
src/ astNodePrint.cpp	
Define the Tang::AstNodePrint class	306
src/ astNodeReturn.cpp	
Define the Tang::AstNodeReturn class	306
src/ astNodeSlice.cpp	
Define the Tang::AstNodeSlice class	307
src/ astNodeString.cpp	
Define the Tang::AstNodeString class	308
src/ astNodeTernary.cpp	
Define the Tang::AstNodeTernary class	309
src/ astNodeUnary.cpp	
Define the Tang::AstNodeUnary class	310
src/ astNodeWhile.cpp	
Define the Tang::AstNodeWhile class	310
src/ computedExpression.cpp	
Define the Tang::ComputedExpression class	311
src/ computedExpressionArray.cpp	
Define the Tang::ComputedExpressionArray class	312
src/ computedExpressionBoolean.cpp	
Define the Tang::ComputedExpressionBoolean class	313
src/ computedExpressionCompiledFunction.cpp	
Define the Tang::ComputedExpressionCompiledFunction class	313

src/computedExpressionError.cpp	
Define the Tang::ComputedExpressionError class	314
src/computedExpressionFloat.cpp	
Define the Tang::ComputedExpressionFloat class	315
src/computedExpressionInteger.cpp	
Define the Tang::ComputedExpressionInteger class	315
src/computedExpressionString.cpp	
Define the Tang::ComputedExpressionString class	316
src/error.cpp	
Define the Tang::Error class	317
src/program-dumpBytecode.cpp	
Define the Tang::Program::dumpBytecode method	318
src/program-execute.cpp	
Define the Tang::Program::execute method	319
src/program.cpp	
Define the Tang::Program class	320
src/tangBase.cpp	
Define the Tang::TangBase class	321
src/unicodeString.cpp	
Contains the function declarations for the Tang::UnicodeString class and the interface to ICU	322
test/test.cpp	
Test the general language behaviors	322
test/testGarbageCollected.cpp	
Test the generic behavior of the Tang::GarbageCollected class	324
test/testSingletonObjectPool.cpp	
Test the generic behavior of the Tang::SingletonObjectPool class	325
test/testUnicodeString.cpp	
Contains tests for the Tang::UnicodeString class	326

Chapter 5

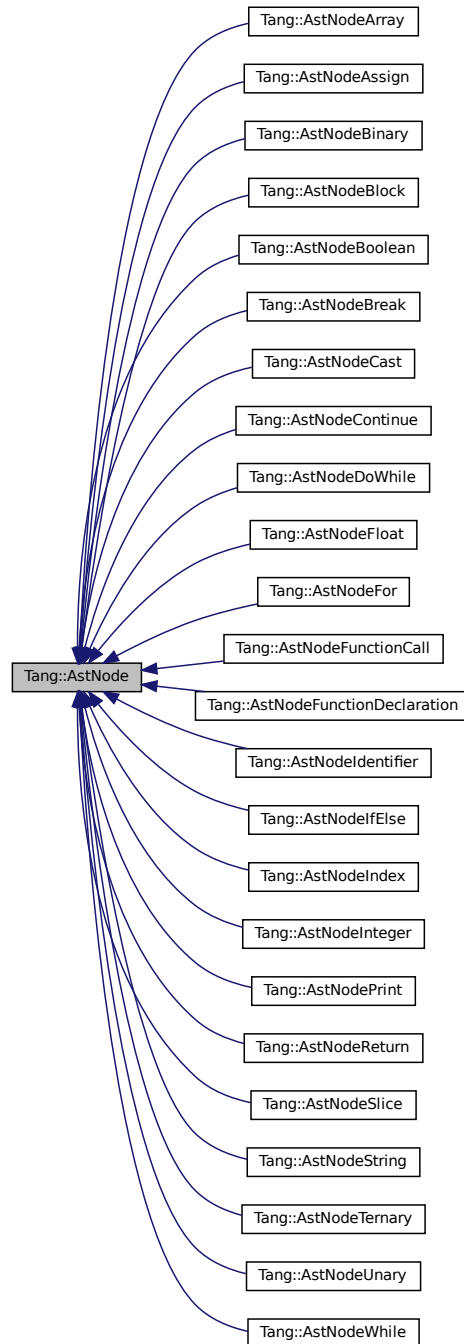
Class Documentation

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 Types

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

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

Public Member Functions

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

5.1.1 Detailed Description

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

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

5.1.2 Member Enumeration Documentation

5.1.2.1 PreprocessState

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

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

Enumerator

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

5.1.3 Constructor & Destructor Documentation

5.1.3.1 AstNode()

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

The generic constructor.

It should never be called on its own.

Parameters

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

5.1.4 Member Function Documentation

5.1.4.1 compile()

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

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

Parameters

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

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

Here is the call graph for this function:



5.1.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.1.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

Here is the call graph for this function:



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

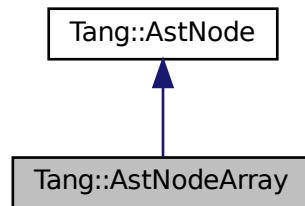
- [include/astNode.hpp](#)
- [src/astNode.cpp](#)

5.2 Tang::AstNodeArray Class Reference

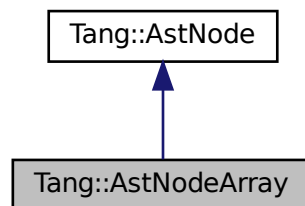
An [AstNode](#) that represents an array literal.

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

Inheritance diagram for Tang::AstNodeArray:



Collaboration diagram for Tang::AstNodeArray:



Public Types

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

Public Member Functions

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

5.2.1 Detailed Description

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

5.2.2 Member Enumeration Documentation

5.2.2.1 PreprocessState

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

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

Enumerator

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

5.2.3 Constructor & Destructor Documentation

5.2.3.1 AstNodeArray()

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

The constructor.

Parameters

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

5.2.4 Member Function Documentation

5.2.4.1 compile()

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

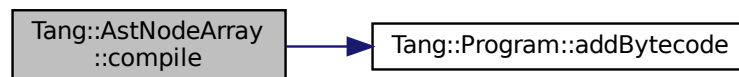
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.2.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.2.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

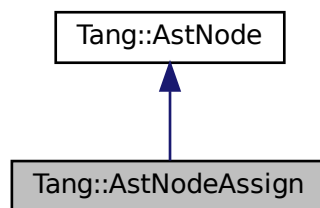
- include/[astNodeArray.hpp](#)
- src/[astNodeArray.cpp](#)

5.3 Tang::AstNodeAssign Class Reference

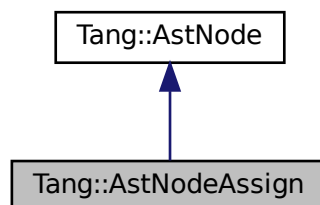
An [AstNode](#) that represents a binary expression.

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

Inheritance diagram for Tang::AstNodeAssign:



Collaboration diagram for Tang::AstNodeAssign:



Public Types

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

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

Public Member Functions

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

5.3.1 Detailed Description

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

5.3.2 Member Enumeration Documentation

5.3.2.1 PreprocessState

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

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

Enumerator

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

5.3.3 Constructor & Destructor Documentation

5.3.3.1 AstNodeAssign()

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

The constructor.

Parameters

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

5.3.4 Member Function Documentation

5.3.4.1 compile()

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

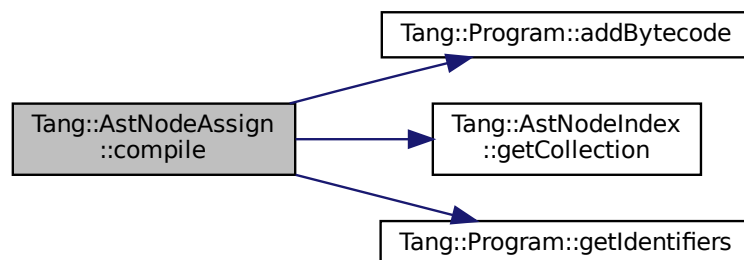
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.3.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.3.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

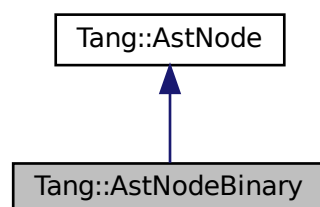
- [include/astNodeAssign.hpp](#)
- [src/astNodeAssign.cpp](#)

5.4 Tang::AstNodeBinary Class Reference

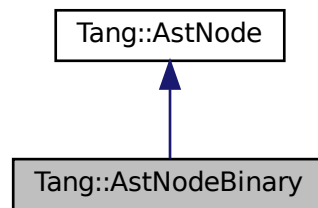
An [AstNode](#) that represents a binary expression.

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

Inheritance diagram for Tang::AstNodeBinary:



Collaboration diagram for Tang::AstNodeBinary:



Public Types

- enum [Operation](#) {
[Add](#) , [Subtract](#) , [Multiply](#) , [Divide](#) ,
[Modulo](#) , [LessThan](#) , [LessThanEqual](#) , [GreaterThan](#) ,
[GreaterThanEqual](#) , [Equal](#) , [NotEqual](#) , [And](#) ,
[Or](#) }
Indicates the type of binary expression that this node represents.
- enum [PreprocessState](#) : int { [Default](#) = 0 , [IsAssignment](#) = 1 }
Bit flags to indicate the state of the preprocess scan as it recursively evaluates the AST.

Public Member Functions

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

5.4.1 Detailed Description

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

5.4.2 Member Enumeration Documentation

5.4.2.1 Operation

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

Indicates the type of binary expression that this node represents.

Enumerator

Add	Indicates lhs + rhs.
Subtract	Indicates lhs - rhs.
Multiply	Indicates lhs * rhs.
Divide	Indicates lhs / rhs.
Modulo	Indicates lhs % rhs.
LessThan	Indicates lhs < rhs.
LessThanEqual	Indicates lhs <= rhs.
GreaterThan	Indicates lhs > rhs.
GreaterThanEqual	Indicates lhs >= rhs.
Equal	Indicates lhs == rhs.
NotEqual	Indicates lhs != rhs.
And	Indicates lhs && rhs with short-circuit evaluation.
Or	Indicates lhs rhs with short-circuit evaluation.

5.4.2.2 PreprocessState

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

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

Enumerator

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

5.4.3 Constructor & Destructor Documentation

5.4.3.1 AstNodeBinary()

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

The constructor.

Parameters

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

5.4.4 Member Function Documentation

5.4.4.1 compile()

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

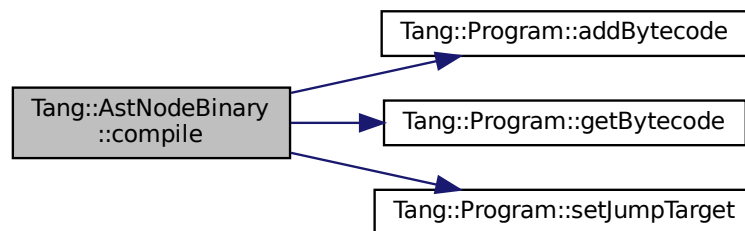
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.4.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.4.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

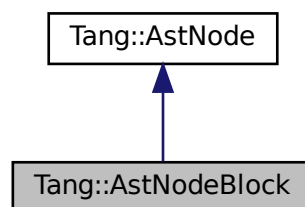
- [include/astNodeBinary.hpp](#)
- [src/astNodeBinary.cpp](#)

5.5 Tang::AstNodeBlock Class Reference

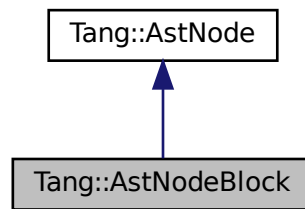
An [AstNode](#) that represents a code block.

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

Inheritance diagram for Tang::AstNodeBlock:



Collaboration diagram for Tang::AstNodeBlock:



Public Types

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

Public Member Functions

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

5.5.1 Detailed Description

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

5.5.2 Member Enumeration Documentation

5.5.2.1 PreprocessState

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

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

Enumerator

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

5.5.3 Constructor & Destructor Documentation

5.5.3.1 AstNodeBlock()

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

The constructor.

Parameters

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

5.5.4 Member Function Documentation

5.5.4.1 compile()

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

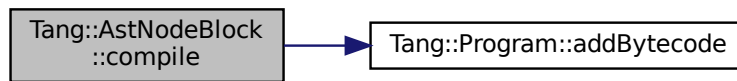
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.5.4.2 compilePreprocess()

```

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.5.4.3 dump()

```

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

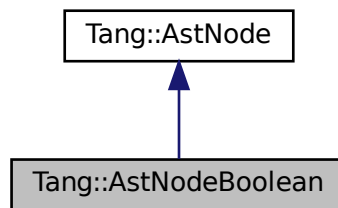
- [include/astNodeBlock.hpp](#)
- [src/astNodeBlock.cpp](#)

5.6 Tang::AstNodeBoolean Class Reference

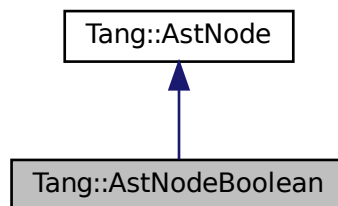
An [AstNode](#) that represents a boolean literal.

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

Inheritance diagram for Tang::AstNodeBoolean:



Collaboration diagram for Tang::AstNodeBoolean:



Public Types

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

Public Member Functions

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

5.6.1 Detailed Description

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

5.6.2 Member Enumeration Documentation

5.6.2.1 PreprocessState

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

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

Enumerator

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

5.6.3 Constructor & Destructor Documentation

5.6.3.1 AstNodeBoolean()

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

The constructor.

Parameters

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

5.6.4 Member Function Documentation

5.6.4.1 compile()

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

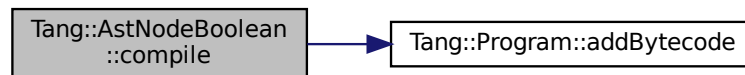
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.6.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.6.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

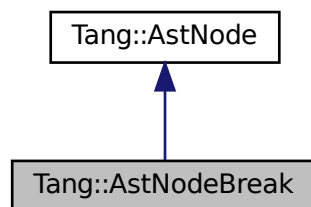
- [include/astNodeBoolean.hpp](#)
- [src/astNodeBoolean.cpp](#)

5.7 Tang::AstNodeBreak Class Reference

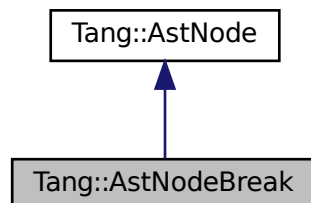
An [AstNode](#) that represents a `break` statement.

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

Inheritance diagram for Tang::AstNodeBreak:



Collaboration diagram for Tang::AstNodeBreak:



Public Types

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

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

Public Member Functions

- [AstNodeBreak](#) ([Tang::location](#) location)

The constructor.

- virtual std::string [dump](#) (std::string indent="") const override

Return a string that describes the contents of the node.

- virtual void [compile](#) ([Tang::Program](#) &program) const override

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

- virtual void [compilePreprocess](#) ([Program](#) &program, [PreprocessState](#) state) const

Run any preprocess analysis needed before compilation.

5.7.1 Detailed Description

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

5.7.2 Member Enumeration Documentation

5.7.2.1 PreprocessState

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

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

Enumerator

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

5.7.3 Constructor & Destructor Documentation

5.7.3.1 AstNodeBreak()

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

The constructor.

Parameters

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

5.7.4 Member Function Documentation

5.7.4.1 compile()

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

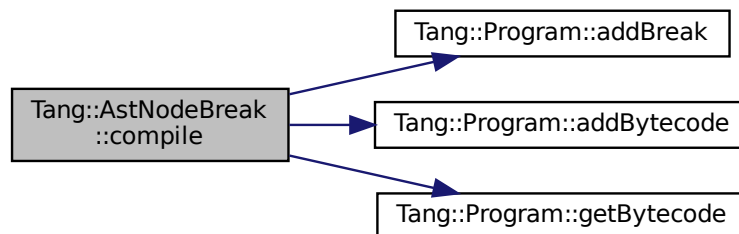
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.7.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.7.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

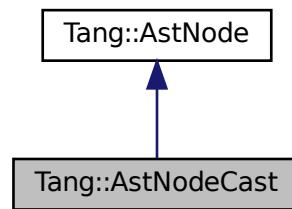
- [include/astNodeBreak.hpp](#)
- [src/astNodeBreak.cpp](#)

5.8 Tang::AstNodeCast Class Reference

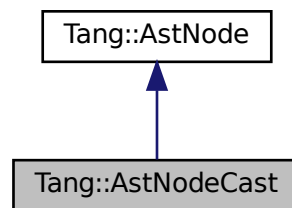
An [AstNode](#) that represents a typecast of an expression.

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

Inheritance diagram for Tang::AstNodeCast:



Collaboration diagram for Tang::AstNodeCast:



Public Types

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

Public Member Functions

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

5.8.1 Detailed Description

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

5.8.2 Member Enumeration Documentation

5.8.2.1 PreprocessState

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

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

Enumerator

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

5.8.2.2 Type

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

The possible types that can be cast to.

Enumerator

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

5.8.3 Constructor & Destructor Documentation

5.8.3.1 AstNodeCast()

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

The constructor.

Parameters

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

5.8.4 Member Function Documentation

5.8.4.1 compile()

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

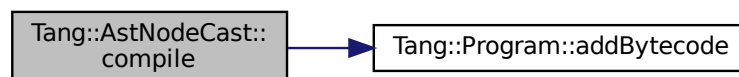
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.8.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.8.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

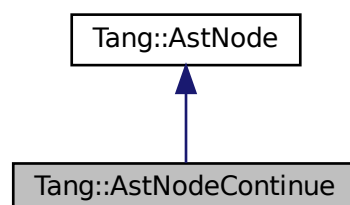
- include/[astNodeCast.hpp](#)
- src/[astNodeCast.cpp](#)

5.9 Tang::AstNodeContinue Class Reference

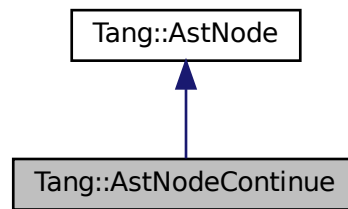
An [AstNode](#) that represents a `continue` statement.

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

Inheritance diagram for Tang::AstNodeContinue:



Collaboration diagram for Tang::AstNodeContinue:



Public Types

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

Public Member Functions

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

5.9.1 Detailed Description

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

5.9.2 Member Enumeration Documentation

5.9.2.1 PreprocessState

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

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

Enumerator

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

5.9.3 Constructor & Destructor Documentation

5.9.3.1 AstNodeContinue()

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

The constructor.

Parameters

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

5.9.4 Member Function Documentation

5.9.4.1 compile()

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

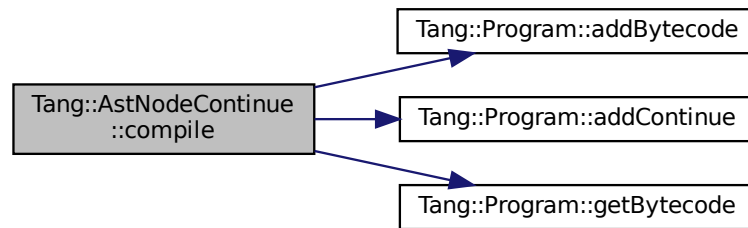
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.9.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.9.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

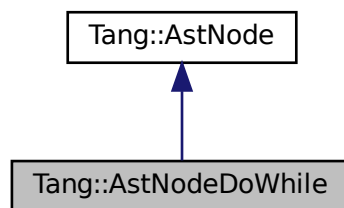
- include/[astNodeContinue.hpp](#)
- src/[astNodeContinue.cpp](#)

5.10 Tang::AstNodeDoWhile Class Reference

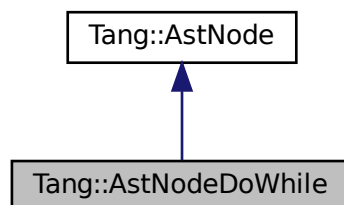
An [AstNode](#) that represents a do..while statement.

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

Inheritance diagram for Tang::AstNodeDoWhile:



Collaboration diagram for Tang::AstNodeDoWhile:



Public Types

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

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

Public Member Functions

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

5.10.1 Detailed Description

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

5.10.2 Member Enumeration Documentation

5.10.2.1 PreprocessState

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

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

Enumerator

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

5.10.3 Constructor & Destructor Documentation

5.10.3.1 AstNodeDoWhile()

```
AstNodeDoWhile::AstNodeDoWhile (
    shared_ptr< AstNode > condition,
    shared_ptr< AstNode > codeBlock,
    Tang::location location )
```

The constructor.

Parameters

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

5.10.4 Member Function Documentation

5.10.4.1 compile()

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

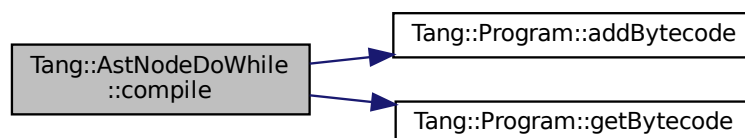
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.10.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.10.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

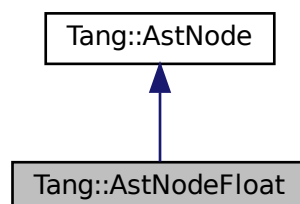
- include/[astNodeDoWhile.hpp](#)
- src/[astNodeDoWhile.cpp](#)

5.11 Tang::AstNodeFloat Class Reference

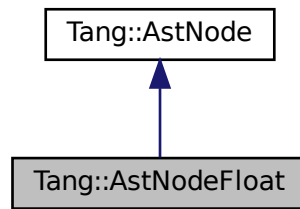
An [AstNode](#) that represents an float literal.

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

Inheritance diagram for Tang::AstNodeFloat:



Collaboration diagram for Tang::AstNodeFloat:



Public Types

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

Public Member Functions

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

5.11.1 Detailed Description

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

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

5.11.2 Member Enumeration Documentation

5.11.2.1 PreprocessState

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

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

Enumerator

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

5.11.3 Constructor & Destructor Documentation

5.11.3.1 AstNodeFloat()

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

The constructor.

Parameters

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

5.11.4 Member Function Documentation

5.11.4.1 compile()

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

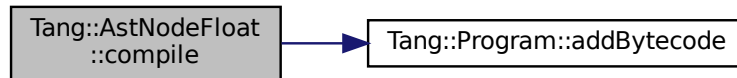
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.11.4.2 compilePreprocess()

```

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.11.4.3 dump()

```

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

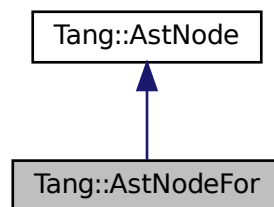
- [include/astNodeFloat.hpp](#)
- [src/astNodeFloat.cpp](#)

5.12 Tang::AstNodeFor Class Reference

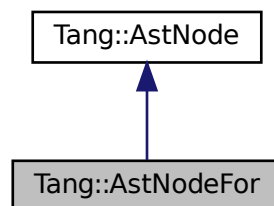
An [AstNode](#) that represents an if() statement.

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

Inheritance diagram for Tang::AstNodeFor:



Collaboration diagram for Tang::AstNodeFor:



Public Types

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

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

Public Member Functions

- [AstNodeFor](#) (shared_ptr< [AstNode](#) > initialization, shared_ptr< [AstNode](#) > condition, shared_ptr< [AstNode](#) > increment, shared_ptr< [AstNode](#) > codeBlock, [Tang::location](#) location)

The constructor.

- virtual std::string [dump](#) (std::string indent="") const override

Return a string that describes the contents of the node.

- virtual void [compile](#) ([Tang::Program](#) &program) const override

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

- virtual void [compilePreprocess](#) ([Program](#) &program, [PreprocessState](#) state) const override

Run any preprocess analysis needed before compilation.

5.12.1 Detailed Description

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

5.12.2 Member Enumeration Documentation

5.12.2.1 PreprocessState

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

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

Enumerator

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

5.12.3 Constructor & Destructor Documentation

5.12.3.1 AstNodeFor()

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

The constructor.

Parameters

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

5.12.4 Member Function Documentation

5.12.4.1 compile()

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

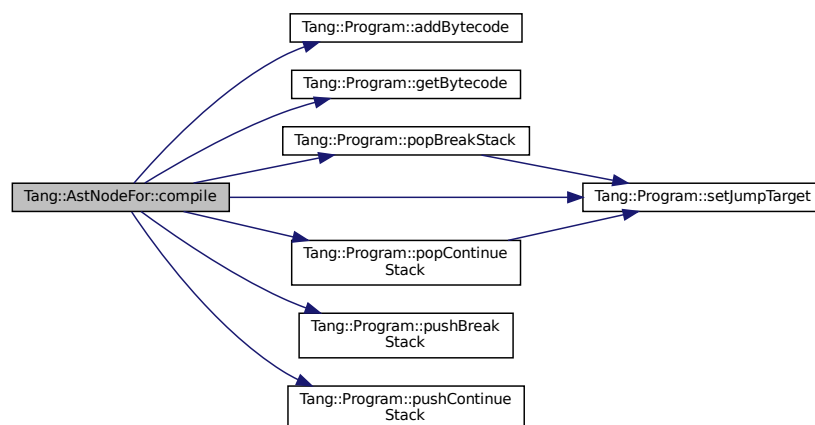
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.12.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.12.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

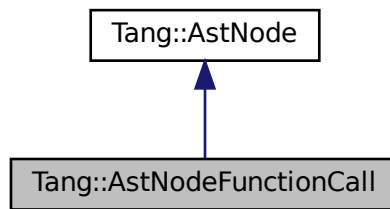
- [include/astNodeFor.hpp](#)
- [src/astNodeFor.cpp](#)

5.13 Tang::AstNodeFunctionCall Class Reference

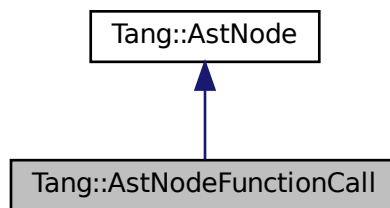
An [AstNode](#) that represents a function call.

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

Inheritance diagram for Tang::AstNodeFunctionCall:



Collaboration diagram for Tang::AstNodeFunctionCall:



Public Types

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

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

Public Member Functions

- [AstNodeFunctionCall](#) (std::shared_ptr< [AstNode](#) > function, std::vector< std::shared_ptr< [AstNode](#) >> argv, [Tang::location](#) location)

The constructor.

- virtual std::string [dump](#) (std::string indent="") const override

Return a string that describes the contents of the node.

- virtual void [compile](#) ([Tang::Program](#) &program) const override

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

- virtual void [compilePreprocess](#) ([Program](#) &program, [PreprocessState](#) state) const override

Run any preprocess analysis needed before compilation.

5.13.1 Detailed Description

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

5.13.2 Member Enumeration Documentation

5.13.2.1 PreprocessState

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

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

Enumerator

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

5.13.3 Constructor & Destructor Documentation

5.13.3.1 AstNodeFunctionCall()

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

The constructor.

Parameters

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

5.13.4 Member Function Documentation

5.13.4.1 compile()

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

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

Parameters

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

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

Here is the call graph for this function:



5.13.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.13.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

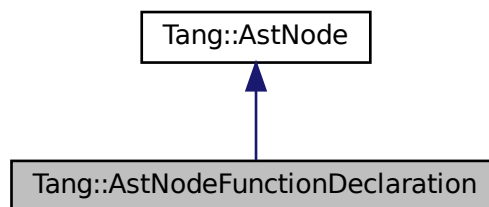
- [include/astNodeFunctionCall.hpp](#)
- [src/astNodeFunctionCall.cpp](#)

5.14 Tang::AstNodeFunctionDeclaration Class Reference

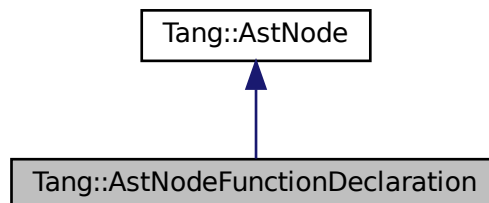
An [AstNode](#) that represents a function declaration.

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

Inheritance diagram for Tang::AstNodeFunctionDeclaration:



Collaboration diagram for Tang::AstNodeFunctionDeclaration:



Public Types

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

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

Public Member Functions

- [AstNodeFunctionDeclaration](#) (std::string name, std::vector< std::string > arguments, shared_ptr< [AstNode](#) > codeBlock, [Tang::location](#) location)

The constructor.

- virtual std::string [dump](#) (std::string indent="") const override

Return a string that describes the contents of the node.

- virtual void [compile](#) ([Tang::Program](#) &program) const override

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

- virtual void [compilePreprocess](#) ([Program](#) &program, [PreprocessState](#) state) const override

Run any preprocess analysis needed before compilation.

5.14.1 Detailed Description

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

5.14.2 Member Enumeration Documentation

5.14.2.1 PreprocessState

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

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

Enumerator

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

5.14.3 Constructor & Destructor Documentation

5.14.3.1 AstNodeFunctionDeclaration()

```
AstNodeFunctionDeclaration::AstNodeFunctionDeclaration (
    std::string name,
```

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

The constructor.

Parameters

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

5.14.4 Member Function Documentation

5.14.4.1 compile()

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

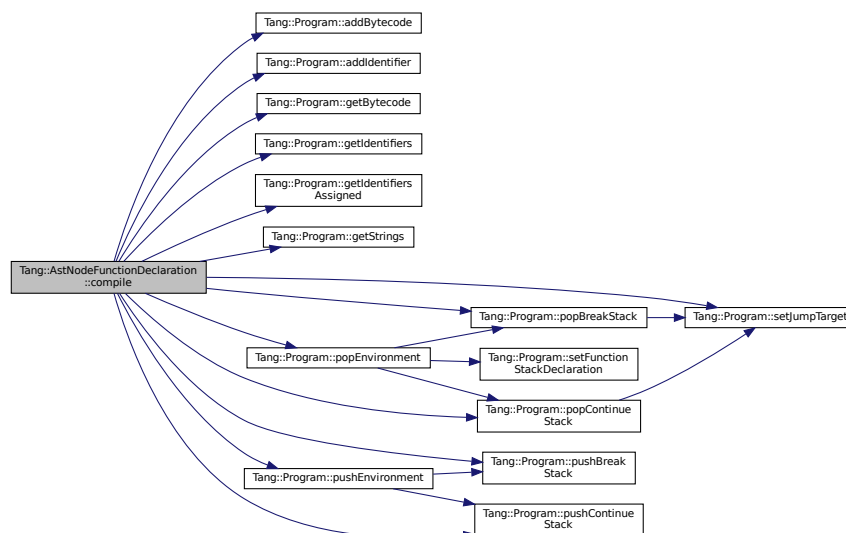
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.14.4.2 compilePreprocess()

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

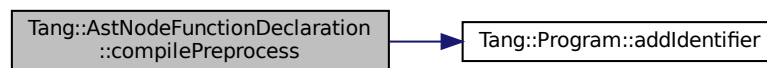
Run any preprocess analysis needed before compilation.

Parameters

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

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

Here is the call graph for this function:



5.14.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

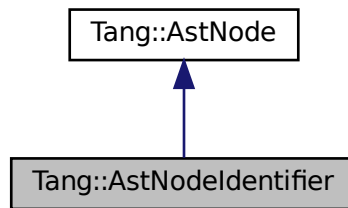
- [include/astNodeFunctionDeclaration.hpp](#)
- [src/astNodeFunctionDeclaration.cpp](#)

5.15 Tang::AstNodeIdentifier Class Reference

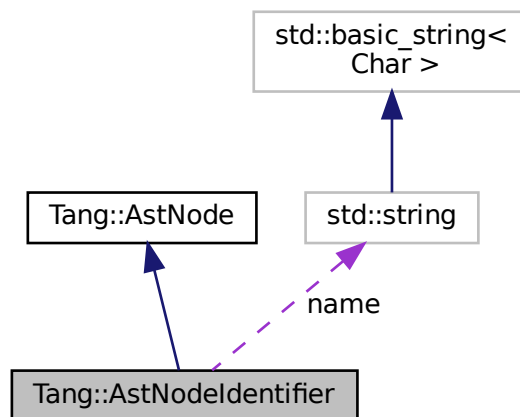
An [AstNode](#) that represents an identifier.

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

Inheritance diagram for Tang::AstNodeIdentifier:



Collaboration diagram for Tang::AstNodeIdentifier:



Public Types

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

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

Public Member Functions

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

Public Attributes

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

5.15.1 Detailed Description

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

Identifier names are represented by a string.

5.15.2 Member Enumeration Documentation

5.15.2.1 PreprocessState

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

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

Enumerator

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

5.15.3 Constructor & Destructor Documentation

5.15.3.1 AstNodeIdentifier()

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

The constructor.

Parameters

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

5.15.4 Member Function Documentation

5.15.4.1 compile()

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

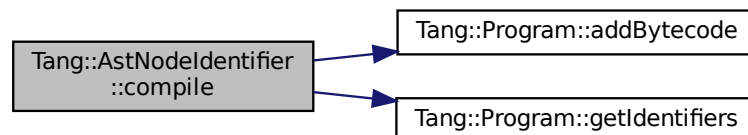
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.15.4.2 compilePreprocess()

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

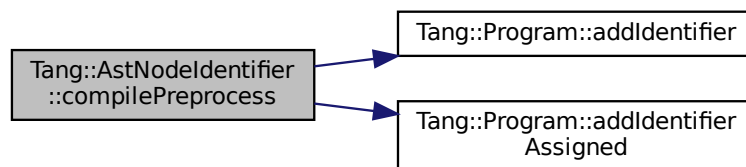
Run any preprocess analysis needed before compilation.

Parameters

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

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

Here is the call graph for this function:



5.15.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

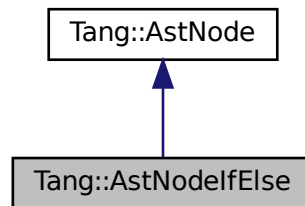
- [include/astNodeIdentifier.hpp](#)
- [src/astNodeIdentifier.cpp](#)

5.16 Tang::AstNodeIfElse Class Reference

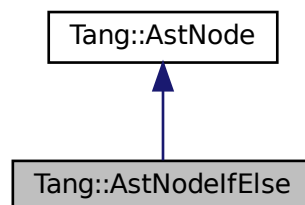
An [AstNode](#) that represents an if..else statement.

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

Inheritance diagram for Tang::AstNodeIfElse:



Collaboration diagram for Tang::AstNodeIfElse:



Public Types

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

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

Public Member Functions

- [AstNodeIfElse](#) (shared_ptr< [AstNode](#) > condition, shared_ptr< [AstNode](#) > thenBlock, shared_ptr< [AstNode](#) > elseBlock, [Tang::location](#) location)

The constructor.

- [AstNodeIfElse](#) (shared_ptr< [AstNode](#) > condition, shared_ptr< [AstNode](#) > thenBlock, [Tang::location](#) location)

The constructor.

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

5.16.1 Detailed Description

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

5.16.2 Member Enumeration Documentation

5.16.2.1 PreprocessState

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

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

Enumerator

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

5.16.3 Constructor & Destructor Documentation

5.16.3.1 AstNodeIfElse() [1/2]

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

The constructor.

Parameters

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

5.16.3.2 AstNodeIfElse() [2/2]

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

The constructor.

Parameters

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

5.16.4 Member Function Documentation

5.16.4.1 compile()

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

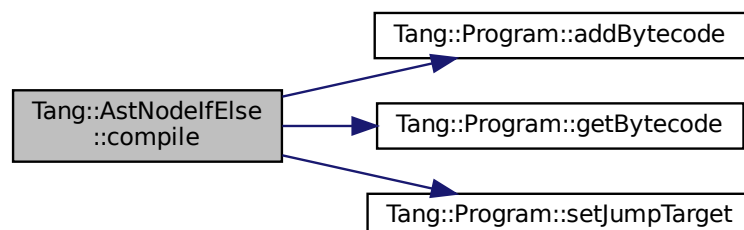
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.16.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.16.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

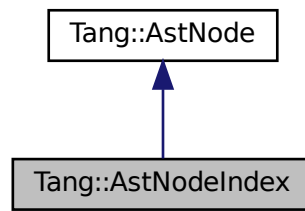
- [include/astNodeIfElse.hpp](#)
- [src/astNodeIfElse.cpp](#)

5.17 Tang::AstNodeIndex Class Reference

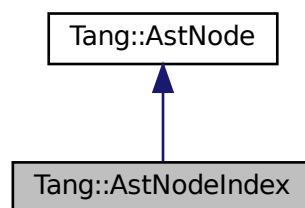
An [AstNode](#) that represents an index into a collection.

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

Inheritance diagram for Tang::AstNodeIndex:



Collaboration diagram for Tang::AstNodeIndex:



Public Types

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

Public Member Functions

- [AstNodeIndex](#) (std::shared_ptr< [AstNode](#) > collection, std::shared_ptr< [AstNode](#) > index, [Tang::location](#) location)
The constructor.
- virtual std::string [dump](#) (std::string indent="") const override
Return a string that describes the contents of the node.
- virtual void [compile](#) ([Tang::Program](#) &program) const override
Compile the ast of the provided [Tang::Program](#).
- virtual void [compilePreprocess](#) ([Program](#) &program, [PreprocessState](#) state) const override
Run any preprocess analysis needed before compilation.
- const std::shared_ptr< const [AstNode](#) > [getCollection](#) () const
Return a shared pointer to the [AstNode](#) serving as the Collection.
- const std::shared_ptr< const [AstNode](#) > [getIndex](#) () const
Return a shared pointer to the [AstNode](#) serving as the Index.

5.17.1 Detailed Description

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

5.17.2 Member Enumeration Documentation

5.17.2.1 PreprocessState

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

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

Enumerator

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

5.17.3 Constructor & Destructor Documentation

5.17.3.1 AstNodeIndex()

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

The constructor.

Parameters

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

5.17.4 Member Function Documentation

5.17.4.1 compile()

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

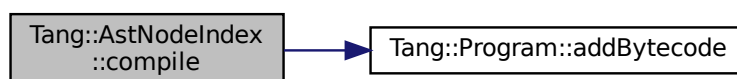
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.17.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.17.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

5.17.4.4 getCollection()

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

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

Returns

The collection into which we will index.

5.17.4.5 getIndex()

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

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

Returns

The index expression.

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

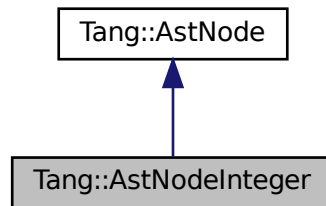
- [include/astNodeIndex.hpp](#)
- [src/astNodeIndex.cpp](#)

5.18 Tang::AstNodeInteger Class Reference

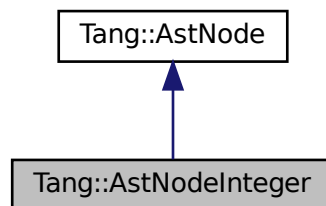
An [AstNode](#) that represents an integer literal.

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

Inheritance diagram for Tang::AstNodeInteger:



Collaboration diagram for Tang::AstNodeInteger:



Public Types

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

Public Member Functions

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

5.18.1 Detailed Description

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

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

5.18.2 Member Enumeration Documentation

5.18.2.1 PreprocessState

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

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

Enumerator

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

5.18.3 Constructor & Destructor Documentation

5.18.3.1 AstNodeInteger()

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

The constructor.

Parameters

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

5.18.4 Member Function Documentation

5.18.4.1 compile()

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

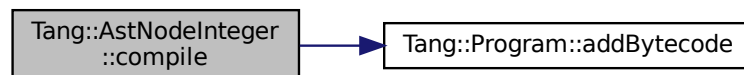
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.18.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.18.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

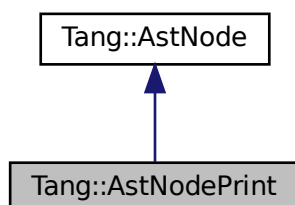
- [include/astNodeInteger.hpp](#)
- [src/astNodeInteger.cpp](#)

5.19 Tang::AstNodePrint Class Reference

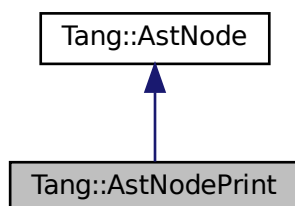
An [AstNode](#) that represents a print typeoperation.

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

Inheritance diagram for Tang::AstNodePrint:



Collaboration diagram for Tang::AstNodePrint:



Public Types

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

Public Member Functions

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

5.19.1 Detailed Description

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

5.19.2 Member Enumeration Documentation

5.19.2.1 PreprocessState

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

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

Enumerator

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

5.19.2.2 Type

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

The type of print() requested.

Enumerator

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

5.19.3 Constructor & Destructor Documentation

5.19.3.1 AstNodePrint()

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

The constructor.

Parameters

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

5.19.4 Member Function Documentation

5.19.4.1 compile()

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

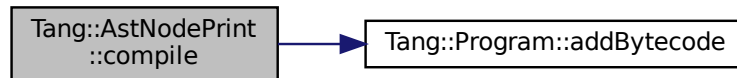
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.19.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.19.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

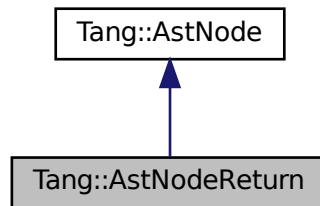
- [include/astNodePrint.hpp](#)
- [src/astNodePrint.cpp](#)

5.20 Tang::AstNodeReturn Class Reference

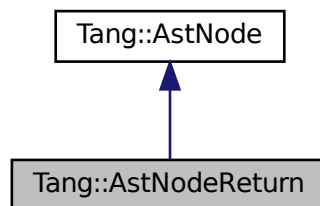
An [AstNode](#) that represents a `return` statement.

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

Inheritance diagram for Tang::AstNodeReturn:



Collaboration diagram for Tang::AstNodeReturn:



Public Types

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

Public Member Functions

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

5.20.1 Detailed Description

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

5.20.2 Member Enumeration Documentation

5.20.2.1 PreprocessState

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

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

Enumerator

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

5.20.3 Constructor & Destructor Documentation

5.20.3.1 AstNodeReturn()

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

The constructor.

Parameters

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

5.20.4 Member Function Documentation

5.20.4.1 compile()

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

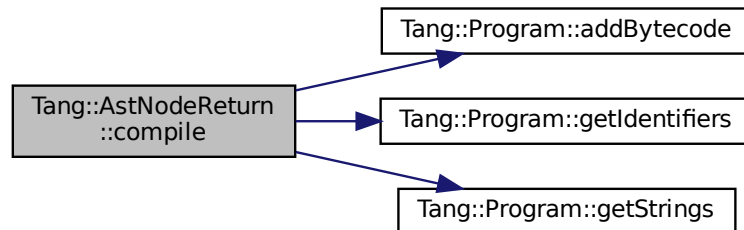
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.20.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.20.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

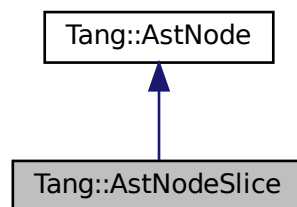
- [include/astNodeReturn.hpp](#)
- [src/astNodeReturn.cpp](#)

5.21 Tang::AstNodeSlice Class Reference

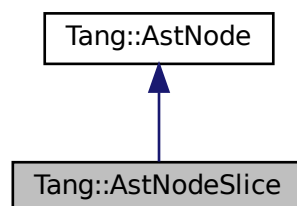
An [AstNode](#) that represents a ternary expression.

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

Inheritance diagram for Tang::AstNodeSlice:



Collaboration diagram for Tang::AstNodeSlice:



Public Types

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

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

Public Member Functions

- [AstNodeSlice](#) (shared_ptr< [AstNode](#) > collection, shared_ptr< [AstNode](#) > begin, shared_ptr< [AstNode](#) > end, shared_ptr< [AstNode](#) > slice, [Tang::location](#) location)

The constructor.

- virtual std::string [dump](#) (std::string indent="") const override

Return a string that describes the contents of the node.

- virtual void [compile](#) ([Tang::Program](#) &program) const override

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

- virtual void [compilePreprocess](#) ([Program](#) &program, [PreprocessState](#) state) const override

Run any preprocess analysis needed before compilation.

5.21.1 Detailed Description

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

5.21.2 Member Enumeration Documentation

5.21.2.1 PreprocessState

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

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

Enumerator

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

5.21.3 Constructor & Destructor Documentation

5.21.3.1 AstNodeSlice()

```
AstNodeSlice::AstNodeSlice (
    shared_ptr< AstNode > collection,
```

```

shared_ptr< AstNode > begin,
shared_ptr< AstNode > end,
shared_ptr< AstNode > slice,
Tang::location location )

```

The constructor.

Parameters

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

5.21.4 Member Function Documentation

5.21.4.1 compile()

```

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

```

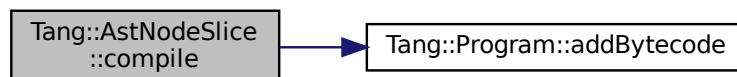
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.21.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.21.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

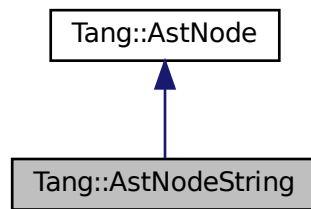
- [include/astNodeSlice.hpp](#)
- [src/astNodeSlice.cpp](#)

5.22 Tang::AstNodeString Class Reference

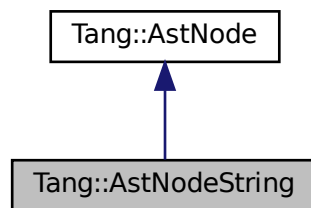
An [AstNode](#) that represents a string literal.

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


Inheritance diagram for Tang::AstNodeString:



Collaboration diagram for Tang::AstNodeString:



Public Types

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

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

Public Member Functions

- [AstNodeString](#) (const string &text, [Tang::location](#) location)
The constructor.
- virtual std::string [dump](#) (std::string indent="") const override
Return a string that describes the contents of the node.
- virtual void [compile](#) ([Tang::Program](#) &program) const override
Compile the ast of the provided Tang::Program.
- virtual void [compilePreprocess](#) ([Program](#) &program, [PreprocessState](#) state) const override
Run any preprocess analysis needed before compilation.
- void [compileLiteral](#) ([Tang::Program](#) &program) const
Compile the string and push it onto the stack.

5.22.1 Detailed Description

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

5.22.2 Member Enumeration Documentation

5.22.2.1 PreprocessState

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

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

Enumerator

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

5.22.3 Constructor & Destructor Documentation

5.22.3.1 AstNodeString()

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

The constructor.

Parameters

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

5.22.4 Member Function Documentation

5.22.4.1 compile()

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

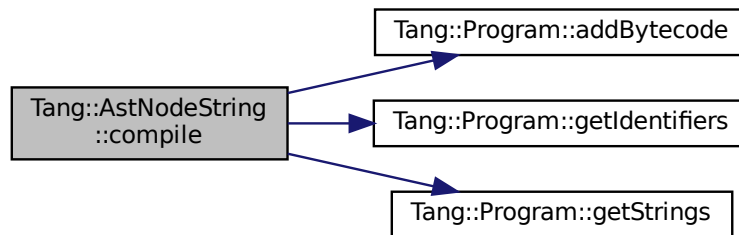
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.22.4.2 compileLiteral()

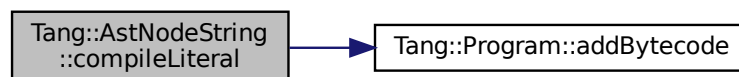
```
void AstNodeString::compileLiteral (
    Tang::Program & program ) const
```

Compile the string and push it onto the stack.

Parameters

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

Here is the call graph for this function:



5.22.4.3 compilePreprocess()

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

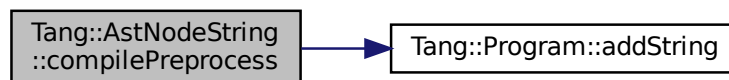
Run any preprocess analysis needed before compilation.

Parameters

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

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

Here is the call graph for this function:



5.22.4.4 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

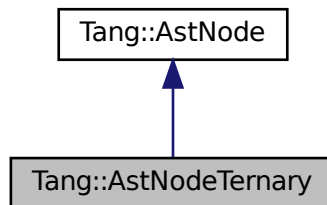
- [include/astNodeString.hpp](#)
- [src/astNodeString.cpp](#)

5.23 Tang::AstNodeTernary Class Reference

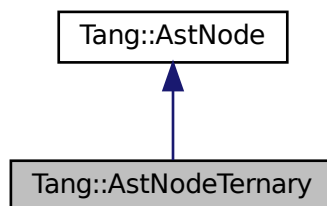
An [AstNode](#) that represents a ternary expression.

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

Inheritance diagram for Tang::AstNodeTernary:



Collaboration diagram for Tang::AstNodeTernary:



Public Types

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

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

Public Member Functions

- [AstNodeTernary](#) (shared_ptr< [AstNode](#) > condition, shared_ptr< [AstNode](#) > trueExpression, shared_ptr< [AstNode](#) > falseExpression, [Tang::location](#) location)

The constructor.

- virtual std::string [dump](#) (std::string indent="") const override

Return a string that describes the contents of the node.

- virtual void [compile](#) ([Tang::Program](#) &program) const override

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

- virtual void [compilePreprocess](#) ([Program](#) &program, [PreprocessState](#) state) const override

Run any preprocess analysis needed before compilation.

5.23.1 Detailed Description

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

5.23.2 Member Enumeration Documentation

5.23.2.1 PreprocessState

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

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

Enumerator

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

5.23.3 Constructor & Destructor Documentation

5.23.3.1 AstNodeTernary()

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

The constructor.

Parameters

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

5.23.4 Member Function Documentation

5.23.4.1 compile()

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

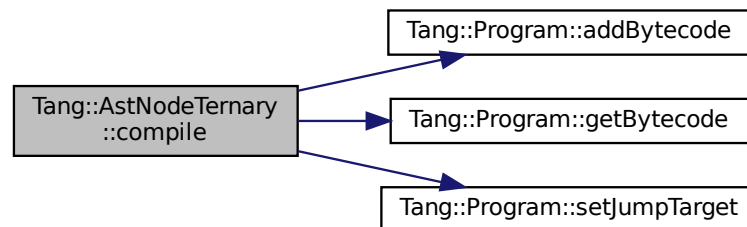
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.23.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.23.4.3 dump()

```
string AstNodeTernary::dump (
```



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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

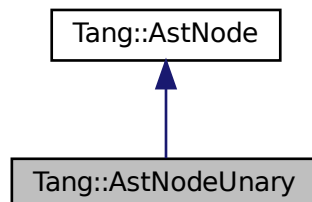
- include/[astNodeTernary.hpp](#)
- src/[astNodeTernary.cpp](#)

5.24 Tang::AstNodeUnary Class Reference

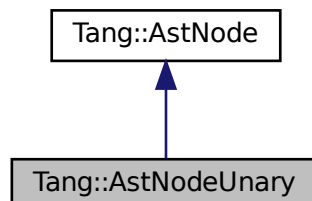
An [AstNode](#) that represents a unary negation.

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

Inheritance diagram for Tang::AstNodeUnary:



Collaboration diagram for Tang::AstNodeUnary:



Public Types

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

Public Member Functions

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

5.24.1 Detailed Description

An `AstNode` that represents a unary negation.

5.24.2 Member Enumeration Documentation

5.24.2.1 Operator

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

The type of operation.

Enumerator

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

5.24.2.2 PreprocessState

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

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

Enumerator

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

5.24.3 Constructor & Destructor Documentation

5.24.3.1 AstNodeUnary()

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

The constructor.

Parameters

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

5.24.4 Member Function Documentation

5.24.4.1 compile()

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

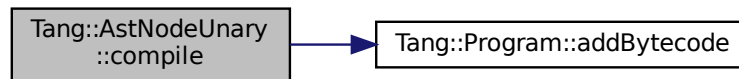
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.24.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.24.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

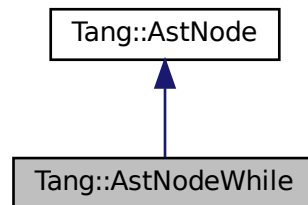
- [include/astNodeUnary.hpp](#)
- [src/astNodeUnary.cpp](#)

5.25 Tang::AstNodeWhile Class Reference

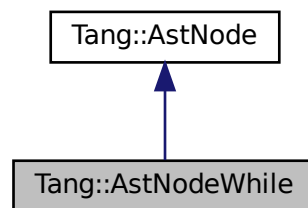
An [AstNode](#) that represents a while statement.

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

Inheritance diagram for Tang::AstNodeWhile:



Collaboration diagram for Tang::AstNodeWhile:



Public Types

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

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

Public Member Functions

- [AstNodeWhile](#) (shared_ptr< [AstNode](#) > condition, shared_ptr< [AstNode](#) > codeBlock, [Tang::location](#) location)

The constructor.

- virtual std::string [dump](#) (std::string indent="") const override

Return a string that describes the contents of the node.

- virtual void [compile](#) ([Tang::Program](#) &program) const override

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

- virtual void [compilePreprocess](#) ([Program](#) &program, [PreprocessState](#) state) const override

Run any preprocess analysis needed before compilation.

5.25.1 Detailed Description

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

5.25.2 Member Enumeration Documentation

5.25.2.1 PreprocessState

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

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

Enumerator

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

5.25.3 Constructor & Destructor Documentation

5.25.3.1 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.

5.25.4 Member Function Documentation

5.25.4.1 compile()

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

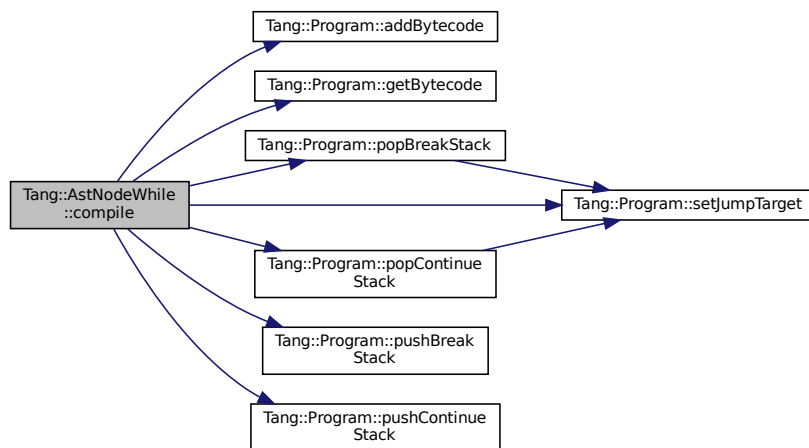
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.25.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.25.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

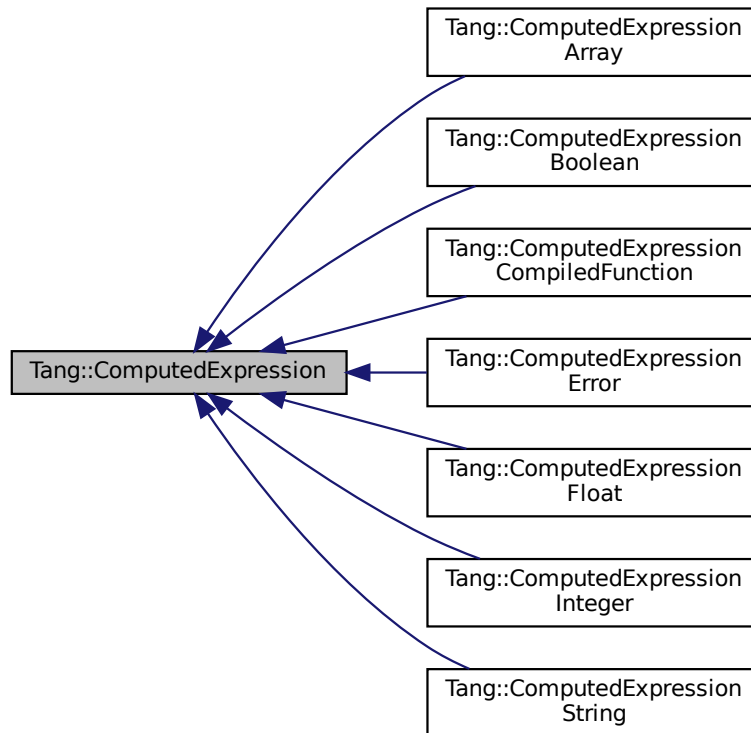
- [include/astNodeWhile.hpp](#)
- [src/astNodeWhile.cpp](#)

5.26 Tang::ComputedExpression Class Reference

Represents the result of a computation that has been executed.

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


Inheritance diagram for Tang::ComputedExpression:



Public Member Functions

- virtual `~ComputedExpression ()`
The object destructor.
- virtual `std::string dump () const`
Output the contents of the `ComputedExpression` as a string.
- virtual `std::string __asCode () const`
Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- virtual `bool isCopyNeeded () const`
Determine whether or not a copy is needed.
- virtual `GarbageCollected makeCopy () const`
Make a copy of the `ComputedExpression` (recursively, if appropriate).
- virtual `bool is_equal (const Tang::integer_t &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal (const Tang::float_t &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal (const bool &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal (const string &val) const`
Check whether or not the computed expression is equal to another value.
- virtual `bool is_equal (const Error &val) const`

- Check whether or not the computed expression is equal to another value.*
- virtual bool `is_equal` (const std::nullptr_t &val) const
- Check whether or not the computed expression is equal to another value.*
- virtual `GarbageCollected __assign_index` (const `GarbageCollected` &index, const `GarbageCollected` &value)
- Perform an index assignment to the supplied value.*
- virtual `GarbageCollected __add` (const `GarbageCollected` &rhs) const
- Compute the result of adding this value and the supplied value.*
- virtual `GarbageCollected __subtract` (const `GarbageCollected` &rhs) const
- Compute the result of subtracting this value and the supplied value.*
- virtual `GarbageCollected __multiply` (const `GarbageCollected` &rhs) const
- Compute the result of multiplying this value and the supplied value.*
- virtual `GarbageCollected __divide` (const `GarbageCollected` &rhs) const
- Compute the result of dividing this value and the supplied value.*
- virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const
- Compute the result of moduloing this value and the supplied value.*
- virtual `GarbageCollected __negative` () const
- Compute the result of negating this value.*
- virtual `GarbageCollected __not` () const
- Compute the logical not of this value.*
- virtual `GarbageCollected __lessThan` (const `GarbageCollected` &rhs) const
- Compute the "less than" comparison.*
- virtual `GarbageCollected __equal` (const `GarbageCollected` &rhs) const
- Perform an equality test.*
- virtual `GarbageCollected __index` (const `GarbageCollected` &index) const
- Perform an index operation.*
- virtual `GarbageCollected __slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const
- Perform a slice operation.*
- virtual `GarbageCollected __integer` () const
- Perform a type cast to integer.*
- virtual `GarbageCollected __float` () const
- Perform a type cast to float.*
- virtual `GarbageCollected __boolean` () const
- Perform a type cast to boolean.*
- virtual `GarbageCollected __string` () const
- Perform a type cast to string.*

5.26.1 Detailed Description

Represents the result of a computation that has been executed.

By default, it will represent a NULL value.

5.26.2 Member Function Documentation

5.26.2.1 `__add()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.26.2.2 __asCode()

```
string ComputedExpression::__asCode ( ) const [virtual]
```

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

Returns

A code-string representation of the computed expression.

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

5.26.2.3 __assign_index()

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.26.2.4 `__boolean()`

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.26.2.5 `__divide()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.26.2.6 `__equal()`

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

Perform an equality test.

Parameters

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

Returns

The result of the the operation.

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

5.26.2.7 __float()

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.26.2.8 __index()

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual]
```

Perform an index operation.

Parameters

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

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.26.2.9 __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](#).

5.26.2.10 __lessThan()

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

Compute the "less than" comparison.

Parameters

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

Returns

The result of the the operation.

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

5.26.2.11 __modulo()

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

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

Parameters

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

Returns

The result of the operation.

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

5.26.2.12 __multiply()

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

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

Parameters

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

Returns

The result of the operation.

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

5.26.2.13 __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](#).

5.26.2.14 __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](#).

5.26.2.15 __slice()

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

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

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.26.2.16 `__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](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.26.2.17 `__subtract()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.26.2.18 dump()

```
string ComputedExpression::dump ( ) const [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

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

5.26.2.19 is_equal() [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.26.2.20 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.26.2.21 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual]
```

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

Parameters

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

Returns

True if equal, false if not.

5.26.2.22 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const string & val ) const [virtual]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.26.2.23 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.26.2.24 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.26.2.25 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

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

5.26.2.26 makeCopy()

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

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

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

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

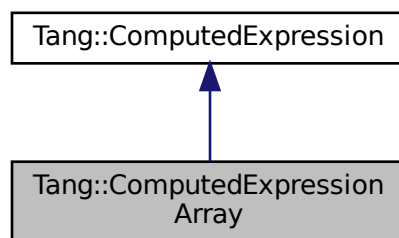
- [include/computedExpression.hpp](#)
- [src/computedExpression.cpp](#)

5.27 Tang::ComputedExpressionArray Class Reference

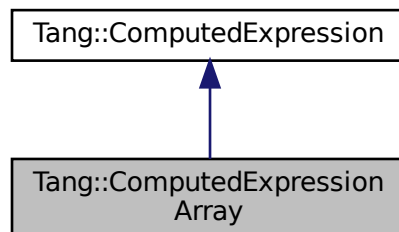
Represents an Array that is the result of a computation.

```
#include <computedExpressionArray.hpp>
```

Inheritance diagram for Tang::ComputedExpressionArray:



Collaboration diagram for Tang::ComputedExpressionArray:



Public Member Functions

- [ComputedExpressionArray](#) (std::vector< [Tang::GarbageCollected](#) > contents)
Construct an Array result.
- virtual std::string [dump](#) () const override
Output the contents of the [ComputedExpression](#) as a string.
- virtual bool [isCopyNeeded](#) () const override
Determine whether or not a copy is needed.
- [GarbageCollected](#) [makeCopy](#) () const override
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual [GarbageCollected](#) [__index](#) (const [GarbageCollected](#) &index) const override
Perform an index operation.
- virtual [GarbageCollected](#) [__slice](#) (const [GarbageCollected](#) &begin, const [GarbageCollected](#) &end, const [GarbageCollected](#) &skip) const override
Perform a slice operation.
- virtual [GarbageCollected](#) [__assign_index](#) (const [GarbageCollected](#) &index, const [GarbageCollected](#) &value) override
Perform an index assignment to the supplied value.
- virtual [GarbageCollected](#) [__string](#) () const override
Perform a type cast to string.
- virtual std::string [__asCode](#) () const
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- virtual bool [is_equal](#) (const [Tang::integer_t](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Tang::float_t](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const bool &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const string &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Error](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected](#) [__add](#) (const [GarbageCollected](#) &rhs) const
Compute the result of adding this value and the supplied value.
- virtual [GarbageCollected](#) [__subtract](#) (const [GarbageCollected](#) &rhs) const
Compute the result of subtracting this value and the supplied value.
- virtual [GarbageCollected](#) [__multiply](#) (const [GarbageCollected](#) &rhs) const
Compute the result of multiplying this value and the supplied value.
- virtual [GarbageCollected](#) [__divide](#) (const [GarbageCollected](#) &rhs) const
Compute the result of dividing this value and the supplied value.
- virtual [GarbageCollected](#) [__modulo](#) (const [GarbageCollected](#) &rhs) const
Compute the result of moduloing this value and the supplied value.
- virtual [GarbageCollected](#) [__negative](#) () const
Compute the result of negating this value.
- virtual [GarbageCollected](#) [__not](#) () const
Compute the logical not of this value.
- virtual [GarbageCollected](#) [__lessThan](#) (const [GarbageCollected](#) &rhs) const
Compute the "less than" comparison.
- virtual [GarbageCollected](#) [__equal](#) (const [GarbageCollected](#) &rhs) const

Perform an equality test.

- virtual `GarbageCollected __integer () const`

Perform a type cast to integer.

- virtual `GarbageCollected __float () const`

Perform a type cast to float.

- virtual `GarbageCollected __boolean () const`

Perform a type cast to boolean.

5.27.1 Detailed Description

Represents an Array that is the result of a computation.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 ComputedExpressionArray()

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

Construct an Array result.

Parameters

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

5.27.3 Member Function Documentation

5.27.3.1 __add()

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

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

Parameters

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

Returns

The result of the operation.

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

5.27.3.2 __asCode()

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

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

Returns

A code-string representation of the computed expression.

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

5.27.3.3 __assign_index()

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.27.3.4 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.27.3.5 __divide()

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

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

Parameters

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

Returns

The result of the operation.

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

5.27.3.6 __equal()

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

Perform an equality test.

Parameters

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

Returns

The result of the the operation.

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

5.27.3.7 __float()

```
GarbageCollected ComputedExpression::__float ( ) const [virtual], [inherited]
```

Perform a type cast to float.

Returns

The result of the the operation.

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

5.27.3.8 __index()

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.27.3.9 __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](#).

5.27.3.10 __lessThan()

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

Compute the "less than" comparison.

Parameters

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

Returns

The result of the the operation.

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

5.27.3.11 `__modulo()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.27.3.12 `__multiply()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.27.3.13 __negative()

`GarbageCollected` ComputedExpression::__negative () const [virtual], [inherited]

Compute the result of negating this value.

Returns

The result of the operation.

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

5.27.3.14 __not()

`GarbageCollected` ComputedExpression::__not () const [virtual], [inherited]

Compute the logical not of this value.

Returns

The result of the operation.

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

5.27.3.15 __slice()

```
GarbageCollected ComputedExpressionArray::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [override], [virtual]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

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

Returns

The result of the operation.

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

5.27.3.16 __string()

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

Perform a type cast to string.

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.27.3.17 __subtract()**

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

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

Parameters

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

Returns

The result of the operation.

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

5.27.3.18 dump()

```
string ComputedExpressionArray::dump ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

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

5.27.3.19 is_equal() [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.27.3.20 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.27.3.21 `is_equal()` [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

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

Parameters

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

Returns

True if equal, false if not.

5.27.3.22 `is_equal()` [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const string & val ) const [virtual], [inherited]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.27.3.23 `is_equal()` [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.27.3.24 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.27.3.25 isCopyNeeded()

```
bool ComputedExpressionArray::isCopyNeeded ( ) const [override], [virtual]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

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

5.27.3.26 makeCopy()

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

Make a copy of the `ComputedExpression` (recursively, if appropriate).

Returns

A `Tang::GarbageCollected` value for the new `ComputedExpression`.

Reimplemented from `Tang::ComputedExpression`.

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

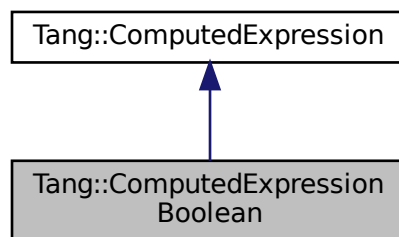
- `include/computedExpressionArray.hpp`
- `src/computedExpressionArray.cpp`

5.28 Tang::ComputedExpressionBoolean Class Reference

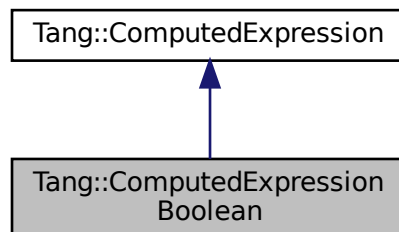
Represents an Boolean that is the result of a computation.

```
#include <computedExpressionBoolean.hpp>
```

Inheritance diagram for `Tang::ComputedExpressionBoolean`:



Collaboration diagram for `Tang::ComputedExpressionBoolean`:



Public Member Functions

- [ComputedExpressionBoolean](#) (bool val)
Construct an Boolean result.
- virtual std::string [dump](#) () const override
Output the contents of the [ComputedExpression](#) as a string.
- [GarbageCollected](#) [makeCopy](#) () const override
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual bool [is_equal](#) (const bool &val) const override
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected](#) [__not](#) () const override
Compute the logical not of this value.
- virtual [GarbageCollected](#) [__equal](#) (const [GarbageCollected](#) &rhs) const override
Perform an equality test.
- virtual [GarbageCollected](#) [__integer](#) () const override
Perform a type cast to integer.
- virtual [GarbageCollected](#) [__float](#) () const override
Perform a type cast to float.
- virtual [GarbageCollected](#) [__boolean](#) () const override
Perform a type cast to boolean.
- virtual std::string [__asCode](#) () const
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- virtual bool [isCopyNeeded](#) () const
Determine whether or not a copy is needed.
- virtual bool [is_equal](#) (const [Tang::integer_t](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Tang::float_t](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const string &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Error](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected](#) [__assign_index](#) (const [GarbageCollected](#) &index, const [GarbageCollected](#) &value)
Perform an index assignment to the supplied value.
- virtual [GarbageCollected](#) [__add](#) (const [GarbageCollected](#) &rhs) const
Compute the result of adding this value and the supplied value.
- virtual [GarbageCollected](#) [__subtract](#) (const [GarbageCollected](#) &rhs) const
Compute the result of subtracting this value and the supplied value.
- virtual [GarbageCollected](#) [__multiply](#) (const [GarbageCollected](#) &rhs) const
Compute the result of multiplying this value and the supplied value.
- virtual [GarbageCollected](#) [__divide](#) (const [GarbageCollected](#) &rhs) const
Compute the result of dividing this value and the supplied value.
- virtual [GarbageCollected](#) [__modulo](#) (const [GarbageCollected](#) &rhs) const
Compute the result of moduloing this value and the supplied value.
- virtual [GarbageCollected](#) [__negative](#) () const
Compute the result of negating this value.
- virtual [GarbageCollected](#) [__lessThan](#) (const [GarbageCollected](#) &rhs) const
Compute the "less than" comparison.
- virtual [GarbageCollected](#) [__index](#) (const [GarbageCollected](#) &index) const

Perform an index operation.

- virtual `GarbageCollected __slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const

Perform a slice operation.

- virtual `GarbageCollected __string` () const

Perform a type cast to string.

5.28.1 Detailed Description

Represents an Boolean that is the result of a computation.

5.28.2 Constructor & Destructor Documentation

5.28.2.1 ComputedExpressionBoolean()

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

Construct an Boolean result.

Parameters

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

5.28.3 Member Function Documentation

5.28.3.1 __add()

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

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

Parameters

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

Returns

The result of the operation.

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

5.28.3.2 __asCode()

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

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

Returns

A code-string representation of the computed expression.

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

5.28.3.3 __assign_index()

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.28.3.4 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.28.3.5 `__divide()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.28.3.6 `__equal()`

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

Perform an equality test.

Parameters

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

Returns

The result of the the operation.

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

5.28.3.7 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.28.3.8 __index()

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

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

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.28.3.9 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.28.3.10 __lessThan()

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

Compute the "less than" comparison.

Parameters

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

Returns

The result of the the operation.

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

5.28.3.11 __modulo()

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

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

Parameters

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

Returns

The result of the operation.

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

5.28.3.12 __multiply()

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

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

Parameters

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

Returns

The result of the operation.

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

5.28.3.13 __negative()

```
GarbageCollected ComputedExpression::__negative ( ) const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

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

5.28.3.14 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.28.3.15 __slice()

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

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

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.28.3.16 __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](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.28.3.17 __subtract()

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

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

Parameters

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

Returns

The result of the operation.

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

5.28.3.18 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](#).

5.28.3.19 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](#).

5.28.3.20 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.28.3.21 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

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

Parameters

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

Returns

True if equal, false if not.

5.28.3.22 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const string & val ) const [virtual], [inherited]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.28.3.23 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.28.3.24 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

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

Parameters

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

Returns

True if equal, false if not.

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

5.28.3.25 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

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

5.28.3.26 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:

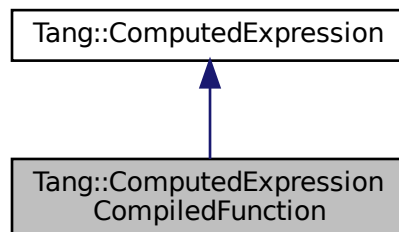
- include/computedExpressionBoolean.hpp
- src/computedExpressionBoolean.cpp

5.29 Tang::ComputedExpressionCompiledFunction Class Reference

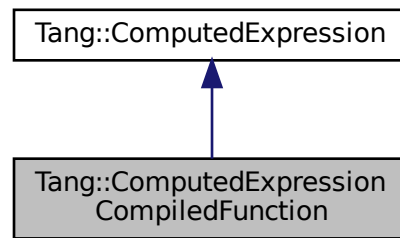
Represents a Compiled Function declared in the script.

```
#include <computedExpressionCompiledFunction.hpp>
```

Inheritance diagram for Tang::ComputedExpressionCompiledFunction:



Collaboration diagram for Tang::ComputedExpressionCompiledFunction:



Public Member Functions

- `ComputedExpressionCompiledFunction` (uint32_t argc, Tang::integer_t pc)
Construct an CompiledFunction.
- virtual std::string `dump` () const override
Output the contents of the `ComputedExpression` as a string.
- `GarbageCollected` `makeCopy` () const override
Make a copy of the `ComputedExpression` (recursively, if appropriate).
- virtual `GarbageCollected` `__equal` (const `GarbageCollected` &rhs) const override
Perform an equality test.
- uint32_t `getArgc` () const
Get the `argc` value.
- Tang::integer_t `getPc` () const
Get the bytecode target.
- virtual std::string `__asCode` () const
Output the contents of the `ComputedExpression` as a string similar to how it would be represented as code.
- virtual bool `isCopyNeeded` () const
Determine whether or not a copy is needed.
- virtual bool `is_equal` (const Tang::integer_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const Tang::float_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const bool &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const string &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const Error &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual `GarbageCollected` `__assign_index` (const `GarbageCollected` &index, const `GarbageCollected` &value)
Perform an index assignment to the supplied value.
- virtual `GarbageCollected` `__add` (const `GarbageCollected` &rhs) const
Compute the result of adding this value and the supplied value.

- virtual [GarbageCollected](#) `__subtract` (const [GarbageCollected](#) &rhs) const
Compute the result of subtracting this value and the supplied value.
- virtual [GarbageCollected](#) `__multiply` (const [GarbageCollected](#) &rhs) const
Compute the result of multiplying this value and the supplied value.
- virtual [GarbageCollected](#) `__divide` (const [GarbageCollected](#) &rhs) const
Compute the result of dividing this value and the supplied value.
- virtual [GarbageCollected](#) `__modulo` (const [GarbageCollected](#) &rhs) const
Compute the result of moduloing this value and the supplied value.
- virtual [GarbageCollected](#) `__negative` () const
Compute the result of negating this value.
- virtual [GarbageCollected](#) `__not` () const
Compute the logical not of this value.
- virtual [GarbageCollected](#) `__lessThan` (const [GarbageCollected](#) &rhs) const
Compute the "less than" comparison.
- virtual [GarbageCollected](#) `__index` (const [GarbageCollected](#) &index) const
Perform an index operation.
- virtual [GarbageCollected](#) `__slice` (const [GarbageCollected](#) &begin, const [GarbageCollected](#) &end, const [GarbageCollected](#) &skip) const
Perform a slice operation.
- virtual [GarbageCollected](#) `__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.

5.29.1 Detailed Description

Represents a Compiled Function declared in the script.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 ComputedExpressionCompiledFunction()

```
ComputedExpressionCompiledFunction::ComputedExpressionCompiledFunction (
    uint32_t argc,
    Tang::integer_t pc )
```

Construct an CompiledFunction.

Parameters

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

5.29.3 Member Function Documentation

5.29.3.1 `__add()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.29.3.2 `__asCode()`

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

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

Returns

A code-string representation of the computed expression.

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

5.29.3.3 `__assign_index()`

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.29.3.4 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.29.3.5 __divide()

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

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

Parameters

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

Returns

The result of the operation.

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

5.29.3.6 __equal()

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

Perform an equality test.

Parameters

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

Returns

The result of the the operation.

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

5.29.3.7 __float()

```
GarbageCollected ComputedExpression::__float ( ) const [virtual], [inherited]
```

Perform a type cast to float.

Returns

The result of the the operation.

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

5.29.3.8 __index()

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

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

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.29.3.9 __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](#).

5.29.3.10 __lessThan()

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

Compute the "less than" comparison.

Parameters

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

Returns

The result of the the operation.

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

5.29.3.11 __modulo()

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

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

Parameters

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

Returns

The result of the operation.

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

5.29.3.12 `__multiply()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.29.3.13 `__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](#).

5.29.3.14 `__not()`

```
GarbageCollected ComputedExpression::__not ( ) const [virtual], [inherited]
```

Compute the logical not of this value.

Returns

The result of the operation.

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

5.29.3.15 __slice()

```

GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]

```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

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

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.29.3.16 __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](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.29.3.17 __subtract()

```

GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]

```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.29.3.18 dump()

```
string ComputedExpressionCompiledFunction::dump ( ) const [override], [virtual]
```

Output the contents of the [ComputedExpression](#) as a string.

Returns

A string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

5.29.3.19 is_equal() [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.29.3.20 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (  
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.29.3.21 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (  
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.29.3.22 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (  
    const string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.29.3.23 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.29.3.24 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.29.3.25 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionArray](#).

5.29.3.26 makeCopy()

`GarbageCollected ComputedExpressionCompiledFunction::makeCopy () const [override], [virtual]`

Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

Returns

A [Tang::GarbageCollected](#) value for the new [ComputedExpression](#).

Reimplemented from [Tang::ComputedExpression](#).

The documentation for this class was generated from the following files:

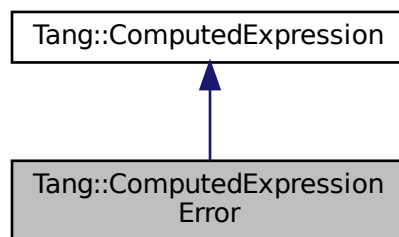
- [include/computedExpressionCompiledFunction.hpp](#)
- [src/computedExpressionCompiledFunction.cpp](#)

5.30 Tang::ComputedExpressionError Class Reference

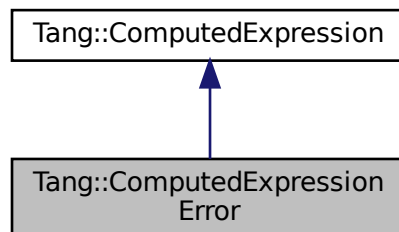
Represents a Runtime [Error](#).

```
#include <computedExpressionError.hpp>
```

Inheritance diagram for Tang::ComputedExpressionError:



Collaboration diagram for Tang::ComputedExpressionError:



Public Member Functions

- [ComputedExpressionError](#) ([Tang::Error](#) error)
Construct a Runtime [Error](#).
- virtual std::string [dump](#) () const override
Output the contents of the [ComputedExpression](#) as a string.
- [GarbageCollected](#) [makeCopy](#) () const override
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual bool [is_equal](#) (const [Error](#) &val) const override
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected](#) [__add](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of adding this value and the supplied value.
- virtual [GarbageCollected](#) [__subtract](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of subtracting this value and the supplied value.
- virtual [GarbageCollected](#) [__multiply](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of multiplying this value and the supplied value.
- virtual [GarbageCollected](#) [__divide](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of dividing this value and the supplied value.
- virtual [GarbageCollected](#) [__modulo](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of moduloing this value and the supplied value.
- virtual [GarbageCollected](#) [__negative](#) () const override
Compute the result of negating this value.
- virtual [GarbageCollected](#) [__not](#) () const override
Compute the logical not of this value.
- virtual [GarbageCollected](#) [__lessThan](#) (const [GarbageCollected](#) &rhs) const override
Compute the "less than" comparison.
- virtual [GarbageCollected](#) [__equal](#) (const [GarbageCollected](#) &rhs) const override
Perform an equality test.
- virtual [GarbageCollected](#) [__integer](#) () const override
Perform a type cast to integer.
- virtual [GarbageCollected](#) [__float](#) () const override
Perform a type cast to float.
- virtual [GarbageCollected](#) [__boolean](#) () const override
Perform a type cast to boolean.
- virtual [GarbageCollected](#) [__string](#) () const override
Perform a type cast to string.
- virtual std::string [__asCode](#) () const
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- virtual bool [isCopyNeeded](#) () const
Determine whether or not a copy is needed.
- virtual bool [is_equal](#) (const [Tang::integer_t](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Tang::float_t](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const bool &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const string &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected](#) [__assign_index](#) (const [GarbageCollected](#) &index, const [GarbageCollected](#) &value)

Perform an index assignment to the supplied value.

- virtual `GarbageCollected __index` (const `GarbageCollected` &index) const

Perform an index operation.

- virtual `GarbageCollected __slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const

Perform a slice operation.

5.30.1 Detailed Description

Represents a Runtime `Error`.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 ComputedExpressionError()

```
ComputedExpressionError::ComputedExpressionError (
    Tang::Error error )
```

Construct a Runtime `Error`.

Parameters

<code>error</code>	The <code>Tang::Error</code> object.
--------------------	--------------------------------------

5.30.3 Member Function Documentation

5.30.3.1 __add()

```
GarbageCollected ComputedExpressionError::__add (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of adding this value and the supplied value.

Parameters

<code>rhs</code>	The <code>GarbageCollected</code> value to add to this.
------------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.2 __asCode()

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.30.3.3 __assign_index()

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionArray](#).

5.30.3.4 __boolean()

```
GarbageCollected ComputedExpressionError::__boolean ( ) const [override], [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.5 `__divide()`

```
GarbageCollected ComputedExpressionError::__divide (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.6 `__equal()`

```
GarbageCollected ComputedExpressionError::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.7 `__float()`

```
GarbageCollected ComputedExpressionError::__float ( ) const [override], [virtual]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.8 __index()

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.30.3.9 __integer()

```
GarbageCollected ComputedExpressionError::__integer ( ) const [override], [virtual]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.10 __lessThan()

```
GarbageCollected ComputedExpressionError::__lessThan (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.11 `__modulo()`

```
GarbageCollected ComputedExpressionError::__modulo (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.12 `__multiply()`

```
GarbageCollected ComputedExpressionError::__multiply (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.13 `__negative()`

```
GarbageCollected ComputedExpressionError::__negative ( ) const [override], [virtual]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.14 __not()

`GarbageCollected` ComputedExpressionError::__not () const [override], [virtual]

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.15 __slice()

`GarbageCollected` ComputedExpression::__slice (
 const `GarbageCollected` & *begin*,
 const `GarbageCollected` & *end*,
 const `GarbageCollected` & *skip*) const [virtual], [inherited]

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.30.3.16 __string()

`GarbageCollected` ComputedExpressionError::__string () const [override], [virtual]

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.17 __subtract()

```
GarbageCollected ComputedExpressionError::__subtract (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.18 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](#).

5.30.3.19 is_equal() [1/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const bool & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionBoolean](#).

5.30.3.20 is_equal() [2/6]

```
bool ComputedExpressionError::is_equal (
    const Error & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.30.3.21 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.30.3.22 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.30.3.23 is_equal() [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.30.3.24 is_equal() [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.30.3.25 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionArray](#).

5.30.3.26 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:

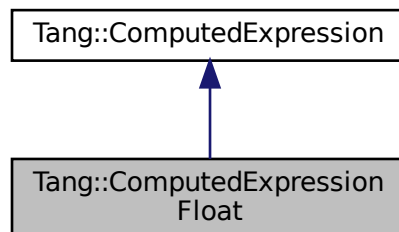
- [include/computedExpressionError.hpp](#)
- [src/computedExpressionError.cpp](#)

5.31 Tang::ComputedExpressionFloat Class Reference

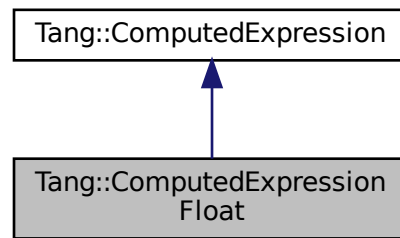
Represents a Float that is the result of a computation.

```
#include <computedExpressionFloat.hpp>
```

Inheritance diagram for Tang::ComputedExpressionFloat:



Collaboration diagram for Tang::ComputedExpressionFloat:



Public Member Functions

- [ComputedExpressionFloat](#) ([Tang::float_t](#) val)
Construct a Float result.
- virtual [std::string dump](#) () const override
Output the contents of the [ComputedExpression](#) as a string.
- [GarbageCollected makeCopy](#) () const override
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual [bool is_equal](#) (const [Tang::integer_t](#) &val) const override
Check whether or not the computed expression is equal to another value.
- virtual [bool is_equal](#) (const [Tang::float_t](#) &val) const override
Check whether or not the computed expression is equal to another value.
- virtual [bool is_equal](#) (const [bool](#) &val) const override
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected __add](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of adding this value and the supplied value.
- virtual [GarbageCollected __subtract](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of subtracting this value and the supplied value.
- virtual [GarbageCollected __multiply](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of multiplying this value and the supplied value.
- virtual [GarbageCollected __divide](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of dividing this value and the supplied value.
- virtual [GarbageCollected __negative](#) () const override
Compute the result of negating this value.
- virtual [GarbageCollected __not](#) () const override
Compute the logical not of this value.
- virtual [GarbageCollected __lessThan](#) (const [GarbageCollected](#) &rhs) const override
Compute the "less than" comparison.
- virtual [GarbageCollected __equal](#) (const [GarbageCollected](#) &rhs) const override
Perform an equality test.
- virtual [GarbageCollected __integer](#) () const override
Perform a type cast to integer.
- virtual [GarbageCollected __float](#) () const override
Perform a type cast to float.

- virtual [GarbageCollected __boolean](#) () const override
Perform a type cast to boolean.
- virtual [GarbageCollected __string](#) () const override
Perform a type cast to string.
- virtual std::string [__asCode](#) () const
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- virtual bool [isCopyNeeded](#) () const
Determine whether or not a copy is needed.
- virtual bool [is_equal](#) (const string &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Error](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected __assign_index](#) (const [GarbageCollected](#) &index, const [GarbageCollected](#) &value)
Perform an index assignment to the supplied value.
- virtual [GarbageCollected __modulo](#) (const [GarbageCollected](#) &rhs) const
Compute the result of moduloing this value and the supplied value.
- virtual [GarbageCollected __index](#) (const [GarbageCollected](#) &index) const
Perform an index operation.
- virtual [GarbageCollected __slice](#) (const [GarbageCollected](#) &begin, const [GarbageCollected](#) &end, const [GarbageCollected](#) &skip) const
Perform a slice operation.

Friends

- class [ComputedExpressionInteger](#)

5.31.1 Detailed Description

Represents a Float that is the result of a computation.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 ComputedExpressionFloat()

```
ComputedExpressionFloat::ComputedExpressionFloat (
    Tang::float_t val )
```

Construct a Float result.

Parameters

<i>val</i>	The float value.
------------	------------------

5.31.3 Member Function Documentation

5.31.3.1 `__add()`

```
GarbageCollected ComputedExpressionFloat::__add (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.2 `__asCode()`

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.31.3.3 `__assign_index()`

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionArray](#).

5.31.3.4 __boolean()

```
GarbageCollected ComputedExpressionFloat::__boolean ( ) const [override], [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.5 __divide()

```
GarbageCollected ComputedExpressionFloat::__divide (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.6 __equal()

```
GarbageCollected ComputedExpressionFloat::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.7 __float()

```
GarbageCollected ComputedExpressionFloat::__float ( ) const [override], [virtual]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.8 __index()

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.31.3.9 __integer()

```
GarbageCollected ComputedExpressionFloat::__integer ( ) const [override], [virtual]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.10 __lessThan()

```
GarbageCollected ComputedExpressionFloat::__lessThan (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.11 __modulo()

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.31.3.12 __multiply()

```
GarbageCollected ComputedExpressionFloat::__multiply (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.13 `__negative()`

```
GarbageCollected ComputedExpressionFloat::__negative ( ) const [override], [virtual]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.14 `__not()`

```
GarbageCollected ComputedExpressionFloat::__not ( ) const [override], [virtual]
```

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.15 `__slice()`

```
GarbageCollected ComputedExpression::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [virtual], [inherited]
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.31.3.16 `__string()`

```
GarbageCollected ComputedExpressionFloat::__string ( ) const [override], [virtual]
```

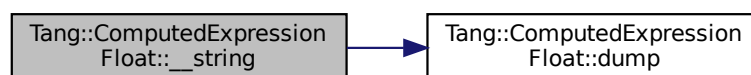
Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:

**5.31.3.17** `__subtract()`

```
GarbageCollected ComputedExpressionFloat::__subtract (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.18 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](#).

5.31.3.19 is_equal() [1/6]

```
bool ComputedExpressionFloat::is_equal (
    const bool & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.20 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.31.3.21 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.31.3.22 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.31.3.23 `is_equal()` [5/6]

```
bool ComputedExpressionFloat::is_equal (
    const Tang::float_t & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.24 `is_equal()` [6/6]

```
bool ComputedExpressionFloat::is_equal (
    const Tang::integer_t & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.31.3.25 `isCopyNeeded()`

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionArray](#).

5.31.3.26 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:

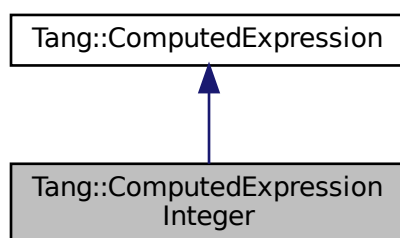
- [include/computedExpressionFloat.hpp](#)
- [src/computedExpressionFloat.cpp](#)

5.32 Tang::ComputedExpressionInteger Class Reference

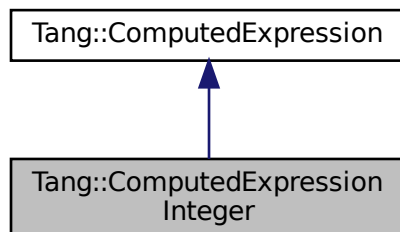
Represents an Integer that is the result of a computation.

```
#include <computedExpressionInteger.hpp>
```

Inheritance diagram for Tang::ComputedExpressionInteger:



Collaboration diagram for Tang::ComputedExpressionInteger:



Public Member Functions

- [ComputedExpressionInteger](#) ([Tang::integer_t](#) val)
Construct an Integer result.
- virtual std::string [dump](#) () const override
Output the contents of the [ComputedExpression](#) as a string.
- [GarbageCollected](#) [makeCopy](#) () const override
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual bool [is_equal](#) (const [Tang::integer_t](#) &val) const override
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Tang::float_t](#) &val) const override
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const bool &val) const override
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected](#) [__add](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of adding this value and the supplied value.
- virtual [GarbageCollected](#) [__subtract](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of subtracting this value and the supplied value.
- virtual [GarbageCollected](#) [__multiply](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of multiplying this value and the supplied value.
- virtual [GarbageCollected](#) [__divide](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of dividing this value and the supplied value.
- virtual [GarbageCollected](#) [__modulo](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of moduloing this value and the supplied value.
- virtual [GarbageCollected](#) [__negative](#) () const override
Compute the result of negating this value.
- virtual [GarbageCollected](#) [__not](#) () const override
Compute the logical not of this value.
- virtual [GarbageCollected](#) [__lessThan](#) (const [GarbageCollected](#) &rhs) const override
Compute the "less than" comparison.
- virtual [GarbageCollected](#) [__equal](#) (const [GarbageCollected](#) &rhs) const override
Perform an equality test.
- virtual [GarbageCollected](#) [__integer](#) () const override
Perform a type cast to integer.
- virtual [GarbageCollected](#) [__float](#) () const override
Perform a type cast to float.
- virtual [GarbageCollected](#) [__boolean](#) () const override
Perform a type cast to boolean.
- virtual [GarbageCollected](#) [__string](#) () const override
Perform a type cast to string.
- virtual std::string [__asCode](#) () const
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- virtual bool [isCopyNeeded](#) () const
Determine whether or not a copy is needed.
- virtual bool [is_equal](#) (const string &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Error](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected](#) [__assign_index](#) (const [GarbageCollected](#) &index, const [GarbageCollected](#) &value)

Perform an index assignment to the supplied value.

- virtual [GarbageCollected](#) `__index` (const [GarbageCollected](#) &index) const

Perform an index operation.

- virtual [GarbageCollected](#) `__slice` (const [GarbageCollected](#) &begin, const [GarbageCollected](#) &end, const [GarbageCollected](#) &skip) const

Perform a slice operation.

Friends

- class **ComputedExpressionFloat**
- class **ComputedExpressionArray**
- class **ComputedExpressionString**

5.32.1 Detailed Description

Represents an Integer that is the result of a computation.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 ComputedExpressionInteger()

```
ComputedExpressionInteger::ComputedExpressionInteger (
    Tang::integer_t val )
```

Construct an Integer result.

Parameters

<i>val</i>	The integer value.
------------	--------------------

5.32.3 Member Function Documentation

5.32.3.1 __add()

```
GarbageCollected ComputedExpressionInteger::__add (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.2 __asCode()

```
string ComputedExpression::__asCode ( ) const [virtual], [inherited]
```

Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented in [Tang::ComputedExpressionString](#).

5.32.3.3 __assign_index()

```
GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionArray](#).

5.32.3.4 __boolean()

```
GarbageCollected ComputedExpressionInteger::__boolean ( ) const [override], [virtual]
```

Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.5 __divide()

```
GarbageCollected ComputedExpressionInteger::__divide (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.6 __equal()

```
GarbageCollected ComputedExpressionInteger::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.7 `__float()`

```
GarbageCollected ComputedExpressionInteger::__float ( ) const [override], [virtual]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.8 `__index()`

```
GarbageCollected ComputedExpression::__index (
    const GarbageCollected & index ) const [virtual], [inherited]
```

Perform an index operation.

Parameters

<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.32.3.9 `__integer()`

```
GarbageCollected ComputedExpressionInteger::__integer ( ) const [override], [virtual]
```

Perform a type cast to integer.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.10 `__lessThan()`

```
GarbageCollected ComputedExpressionInteger::__lessThan (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.11 __modulo()

```
GarbageCollected ComputedExpressionInteger::__modulo (  
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.12 __multiply()

```
GarbageCollected ComputedExpressionInteger::__multiply (  
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.13 __negative()

`GarbageCollected ComputedExpressionInteger::__negative () const [override], [virtual]`

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.14 __not()

`GarbageCollected ComputedExpressionInteger::__not () const [override], [virtual]`

Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.15 __slice()

`GarbageCollected ComputedExpression::__slice (`
 `const GarbageCollected & begin,`
 `const GarbageCollected & end,`
 `const GarbageCollected & skip) const [virtual], [inherited]`

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionString](#), and [Tang::ComputedExpressionArray](#).

5.32.3.16 __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:



5.32.3.17 __subtract()

```
GarbageCollected ComputedExpressionInteger::__subtract (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.18 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](#).

5.32.3.19 is_equal() [1/6]

```
bool ComputedExpressionInteger::is_equal (
    const bool & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.20 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.32.3.21 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.32.3.22 is_equal() [4/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const string & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionString](#).

5.32.3.23 is_equal() [5/6]

```
bool ComputedExpressionInteger::is_equal (
    const Tang::float_t & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.24 is_equal() [6/6]

```
bool ComputedExpressionInteger::is_equal (
    const Tang::integer_t & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.32.3.25 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionArray](#).

5.32.3.26 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:

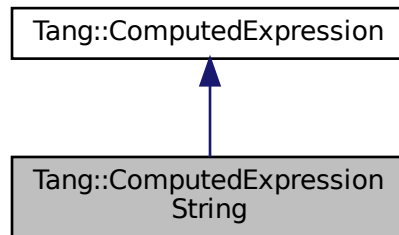
- include/computedExpressionInteger.hpp
- src/computedExpressionInteger.cpp

5.33 Tang::ComputedExpressionString Class Reference

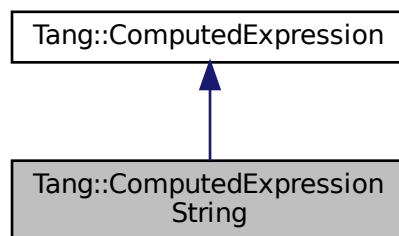
Represents a String that is the result of a computation.

```
#include <computedExpressionString.hpp>
```

Inheritance diagram for Tang::ComputedExpressionString:



Collaboration diagram for Tang::ComputedExpressionString:



Public Member Functions

- [ComputedExpressionString](#) (std::string val)
Construct a String result.
- virtual std::string [dump](#) () const override
Output the contents of the [ComputedExpression](#) as a string.
- virtual std::string [__asCode](#) () const override
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- [GarbageCollected](#) [makeCopy](#) () const override
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual bool [is_equal](#) (const bool &val) const override
Check whether or not the computed expression is equal to another value.

- virtual bool `is_equal` (const string &val) const override
Check whether or not the computed expression is equal to another value.
- virtual `GarbageCollected __index` (const `GarbageCollected` &index) const override
Perform an index operation.
- virtual `GarbageCollected __slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const override
Perform a slice operation.
- virtual `GarbageCollected __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 `isCopyNeeded` () const
Determine whether or not a copy is needed.
- virtual bool `is_equal` (const `Tang::integer_t` &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const `Tang::float_t` &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const `Error` &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual `GarbageCollected __assign_index` (const `GarbageCollected` &index, const `GarbageCollected` &value)
Perform an index assignment to the supplied value.
- virtual `GarbageCollected __subtract` (const `GarbageCollected` &rhs) const
Compute the result of subtracting this value and the supplied value.
- virtual `GarbageCollected __multiply` (const `GarbageCollected` &rhs) const
Compute the result of multiplying this value and the supplied value.
- virtual `GarbageCollected __divide` (const `GarbageCollected` &rhs) const
Compute the result of dividing this value and the supplied value.
- virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __negative` () const
Compute the result of negating this value.
- virtual `GarbageCollected __integer` () const
Perform a type cast to integer.
- virtual `GarbageCollected __float` () const
Perform a type cast to float.

5.33.1 Detailed Description

Represents a String that is the result of a computation.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 ComputedExpressionString()

```
ComputedExpressionString::ComputedExpressionString (
    std::string val )
```

Construct a String result.

Parameters

<i>val</i>	The string value.
------------	-------------------

5.33.3 Member Function Documentation

5.33.3.1 __add()

```
GarbageCollected ComputedExpressionString::__add (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the result of adding this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.33.3.2 __asCode()

```
string ComputedExpressionString::__asCode ( ) const [override], [virtual]
```

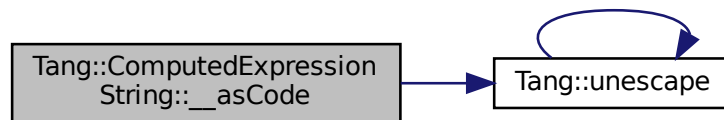
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:

**5.33.3.3 __assign_index()**

```

GarbageCollected ComputedExpression::__assign_index (
    const GarbageCollected & index,
    const GarbageCollected & value ) [virtual], [inherited]
  
```

Perform an index assignment to the supplied value.

Parameters

<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionArray](#).

5.33.3.4 __boolean()

```

GarbageCollected ComputedExpressionString::__boolean ( ) const [override], [virtual]
  
```

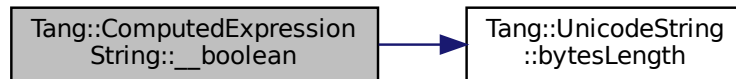
Perform a type cast to boolean.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:

**5.33.3.5 __divide()**

```

GarbageCollected ComputedExpression::__divide (
    const GarbageCollected & rhs ) const [virtual], [inherited]
  
```

Compute the result of dividing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.33.3.6 __equal()

```

GarbageCollected ComputedExpressionString::__equal (
    const GarbageCollected & rhs ) const [override], [virtual]
  
```

Perform an equality test.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.33.3.7 __float()

```
GarbageCollected ComputedExpression::__float ( ) const [virtual], [inherited]
```

Perform a type cast to float.

Returns

The result of the the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionBoolean](#).

5.33.3.8 __index()

```
GarbageCollected ComputedExpressionString::__index (
    const GarbageCollected & index ) const [override], [virtual]
```

Perform an index operation.

Parameters

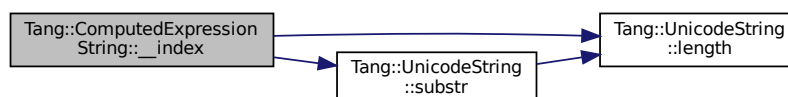
<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.33.3.9 __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](#).

5.33.3.10 __lessThan()

```
GarbageCollected ComputedExpressionString::__lessThan (
    const GarbageCollected & rhs ) const [override], [virtual]
```

Compute the "less than" comparison.

Parameters

<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.33.3.11 __modulo()

```
GarbageCollected ComputedExpression::__modulo (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of moduloing this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionError](#).

5.33.3.12 `__multiply()`

```
GarbageCollected ComputedExpression::__multiply (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of multiplying this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.33.3.13 `__negative()`

```
GarbageCollected ComputedExpression::__negative ( ) const [virtual], [inherited]
```

Compute the result of negating this value.

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.33.3.14 `__not()`

```
GarbageCollected ComputedExpressionString::__not ( ) const [override], [virtual]
```

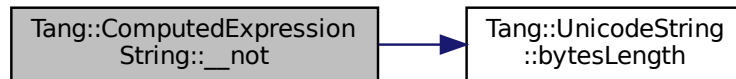
Compute the logical not of this value.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:

**5.33.3.15 __slice()**

```

GarbageCollected ComputedExpressionString::__slice (
    const GarbageCollected & begin,
    const GarbageCollected & end,
    const GarbageCollected & skip ) const [override], [virtual]
  
```

Perform a slice operation.

Convention will follow Python semantics, in which a slice will start at the provided index position, and go up to but not including the end index. The slice will default to an index increment of 1, but can be defined as another integer value.

Parameters

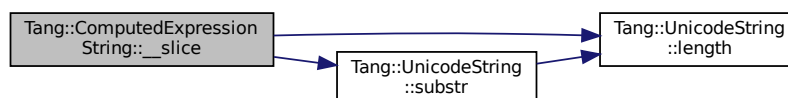
<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

Reimplemented from [Tang::ComputedExpression](#).

Here is the call graph for this function:



5.33.3.16 __string()

```
GarbageCollected ComputedExpressionString::__string ( ) const [override], [virtual]
```

Perform a type cast to string.

Returns

The result of the the operation.

Reimplemented from [Tang::ComputedExpression](#).

5.33.3.17 __subtract()

```
GarbageCollected ComputedExpression::__subtract (
    const GarbageCollected & rhs ) const [virtual], [inherited]
```

Compute the result of subtracting this value and the supplied value.

Parameters

<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

Reimplemented in [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), and [Tang::ComputedExpressionError](#).

5.33.3.18 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](#).

5.33.3.19 is_equal() [1/6]

```
bool ComputedExpressionString::is_equal (
    const bool & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.33.3.20 is_equal() [2/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Error & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionError](#).

5.33.3.21 is_equal() [3/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const std::nullptr_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

5.33.3.22 `is_equal()` [4/6]

```
bool ComputedExpressionString::is_equal (
    const string & val ) const [override], [virtual]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented from [Tang::ComputedExpression](#).

5.33.3.23 `is_equal()` [5/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::float_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.33.3.24 `is_equal()` [6/6]

```
virtual bool Tang::ComputedExpression::is_equal (
    const Tang::integer_t & val ) const [virtual], [inherited]
```

Check whether or not the computed expression is equal to another value.

Parameters

<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

Reimplemented in [Tang::ComputedExpressionInteger](#), and [Tang::ComputedExpressionFloat](#).

5.33.3.25 isCopyNeeded()

```
bool ComputedExpression::isCopyNeeded ( ) const [virtual], [inherited]
```

Determine whether or not a copy is needed.

Copying is only required for ComputedExpressions which serve as containers, such as [ComputedExpressionArray](#) and [ComputedExpressionObject](#).

Returns

Whether or not a copy is needed.

Reimplemented in [Tang::ComputedExpressionArray](#).

5.33.3.26 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:

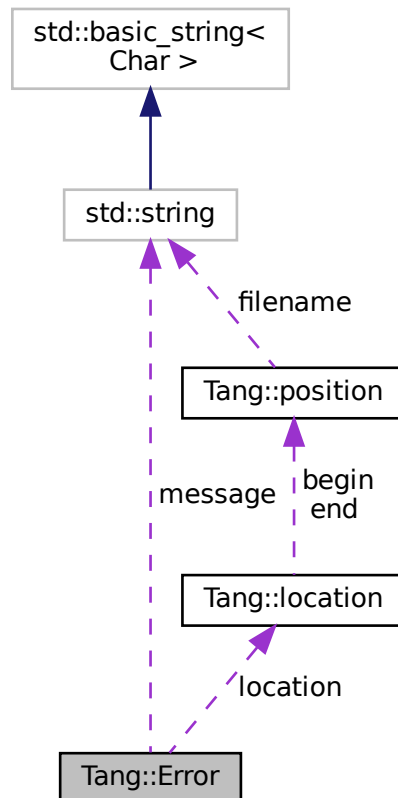
- [include/computedExpressionString.hpp](#)
- [src/computedExpressionString.cpp](#)

5.34 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.

5.34.1 Detailed Description

The [Error](#) class is used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 [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.
----------------	--------------------------------

5.34.2.2 [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.

5.34.3 Friends And Related Function Documentation

5.34.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & out,
    const Error & error ) [friend]
```

Add friendly output.

Parameters

<i>out</i>	The output stream.
<i>error</i>	The Error object.

Returns

The output stream.

The documentation for this class was generated from the following files:

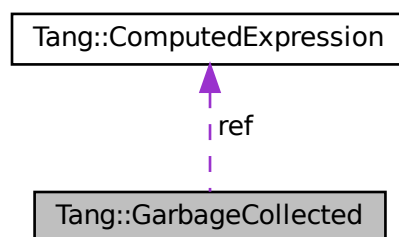
- include/[error.hpp](#)
- src/[error.cpp](#)

5.35 Tang::GarbageCollected Class Reference

A container that acts as a resource-counting garbage collector for the specified type.

```
#include <garbageCollected.hpp>
```

Collaboration diagram for Tang::GarbageCollected:



Public Member Functions

- [GarbageCollected](#) (const [GarbageCollected](#) &other)
Copy Constructor.
- [GarbageCollected](#) ([GarbageCollected](#) &&other)
Move Constructor.
- [GarbageCollected](#) & operator= (const [GarbageCollected](#) &other)
Copy Assignment.
- [GarbageCollected](#) & operator= ([GarbageCollected](#) &&other)
Move Assignment.
- [~GarbageCollected](#) ()
Destructor.
- bool [isCopyNeeded](#) () const
Determine whether or not a copy is needed as determined by the referenced [ComputedExpression](#).
- [GarbageCollected](#) [makeCopy](#) () const
Create a separate copy of the original [GarbageCollected](#) value.
- [ComputedExpression](#) * operator-> () const
Access the tracked object as a pointer.
- [ComputedExpression](#) & operator* () const
Access the tracked object.
- bool operator== (const [Tang::integer_t](#) &val) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- bool operator== (const [Tang::float_t](#) &val) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- bool operator== (const bool &val) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- bool operator== (const std::string &val) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- bool operator== (const char *const &val) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- bool operator== (const [Error](#) &val) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- bool operator== (const std::nullptr_t &null) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- [GarbageCollected](#) operator+ (const [GarbageCollected](#) &rhs) const
Perform an addition between two [GarbageCollected](#) values.
- [GarbageCollected](#) operator- (const [GarbageCollected](#) &rhs) const
Perform a subtraction between two [GarbageCollected](#) values.
- [GarbageCollected](#) operator* (const [GarbageCollected](#) &rhs) const
Perform a multiplication between two [GarbageCollected](#) values.
- [GarbageCollected](#) operator/ (const [GarbageCollected](#) &rhs) const
Perform a division between two [GarbageCollected](#) values.
- [GarbageCollected](#) operator% (const [GarbageCollected](#) &rhs) const
Perform a modulo between two [GarbageCollected](#) values.
- [GarbageCollected](#) operator- () const
Perform a negation on the [GarbageCollected](#) value.
- [GarbageCollected](#) operator! () const
Perform a logical not on the [GarbageCollected](#) value.
- [GarbageCollected](#) operator< (const [GarbageCollected](#) &rhs) const
Perform a < between two [GarbageCollected](#) values.
- [GarbageCollected](#) operator<= (const [GarbageCollected](#) &rhs) const

- Perform $a \leq$ 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.

5.35.1 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.

5.35.2 Constructor & Destructor Documentation

5.35.2.1 *GarbageCollected*() [1/3]

```
GarbageCollected::GarbageCollected (
    const GarbageCollected & other )
```

Copy Constructor.

Parameters

<i>The</i>	other GarbageCollected object to copy.
------------	--

5.35.2.2 GarbageCollected() [2/3]

```
GarbageCollected::GarbageCollected (
    GarbageCollected && other )
```

Move Constructor.

Parameters

<i>The</i>	other GarbageCollected object to move.
------------	--

5.35.2.3 ~GarbageCollected()

```
GarbageCollected::~~GarbageCollected ( )
```

Destructor.

Clean up the tracked object, if appropriate.

5.35.2.4 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.
-----------------	---

5.35.3 Member Function Documentation**5.35.3.1 isCopyNeeded()**

```
bool GarbageCollected::isCopyNeeded ( ) const
```

Determine whether or not a copy is needed as determined by the referenced [ComputedExpression](#).

Returns

Whether or not a copy is needed.

5.35.3.2 make()

```
template<class T , typename... Args>
static GarbageCollected Tang::GarbageCollected::make (
    Args... args ) [inline], [static]
```

Creates a garbage-collected object of the specified type.

Parameters

<i>variable</i>	The arguments to pass to the constructor of the specified type.
-----------------	---

Returns

A [GarbageCollected](#) object.

Here is the call graph for this function:



5.35.3.3 makeCopy()

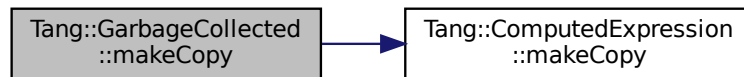
```
GarbageCollected GarbageCollected::makeCopy ( ) const
```

Create a separate copy of the original [GarbageCollected](#) value.

Returns

A [GarbageCollected](#) copy of the original value.

Here is the call graph for this function:

**5.35.3.4 operator"!")()**

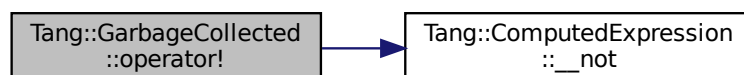
```
GarbageCollected GarbageCollected::operator! ( ) const
```

Perform a logical not on the [GarbageCollected](#) value.

Returns

The result of the operation.

Here is the call graph for this function:

**5.35.3.5 operator"!=(())**

```
GarbageCollected GarbageCollected::operator!= (
    const GarbageCollected & rhs ) const
```

Perform a != between two [GarbageCollected](#) values.

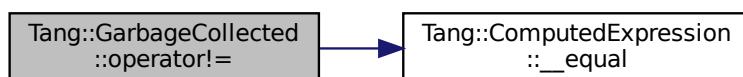
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:

**5.35.3.6 operator%()**

```

GarbageCollected GarbageCollected::operator% (
    const GarbageCollected & rhs ) const
  
```

Perform a modulo between two [GarbageCollected](#) values.

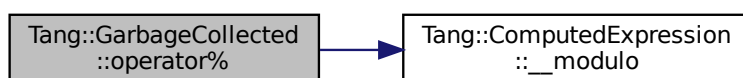
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.35.3.7 operator*() [1/2]

```
ComputedExpression & GarbageCollected::operator* ( ) const
```

Access the tracked object.

Returns

A reference to the tracked object.

5.35.3.8 operator*() [2/2]

```
GarbageCollected GarbageCollected::operator* (
    const GarbageCollected & rhs ) const
```

Perform a multiplication between two [GarbageCollected](#) values.

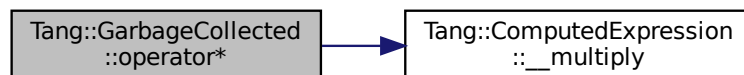
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:

**5.35.3.9 operator+()**

```
GarbageCollected GarbageCollected::operator+ (
    const GarbageCollected & rhs ) const
```

Perform an addition between two [GarbageCollected](#) values.

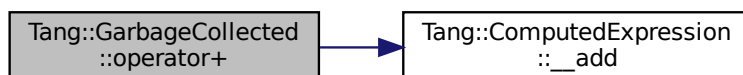
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:

5.35.3.10 `operator-()` [1/2]

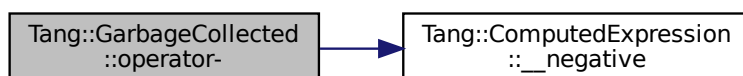
```
GarbageCollected GarbageCollected::operator- ( ) const
```

Perform a negation on the `GarbageCollected` value.

Returns

The result of the operation.

Here is the call graph for this function:

5.35.3.11 `operator-()` [2/2]

```
GarbageCollected GarbageCollected::operator- (
    const GarbageCollected & rhs ) const
```

Perform a subtraction between two `GarbageCollected` values.

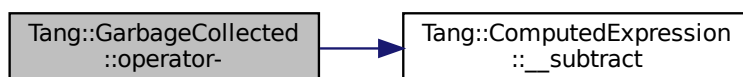
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.35.3.12 operator->()

```
ComputedExpression * GarbageCollected::operator-> ( ) const
```

Access the tracked object as a pointer.

Returns

A pointer to the tracked object.

5.35.3.13 operator/()

```
GarbageCollected GarbageCollected::operator/ (
    const GarbageCollected & rhs ) const
```

Perform a division between two [GarbageCollected](#) values.

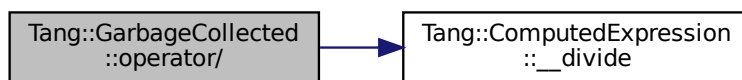
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.35.3.14 `operator<()`

```
GarbageCollected GarbageCollected::operator< (
    const GarbageCollected & rhs ) const
```

Perform a `<` between two `GarbageCollected` values.

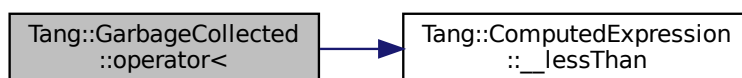
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.35.3.15 `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.

5.35.3.16 operator=() [1/2]

```
GarbageCollected & GarbageCollected::operator= (  
    const GarbageCollected & other )
```

Copy Assignment.

Parameters

<i>The</i>	other GarbageCollected object.
------------	--

5.35.3.17 operator=() [2/2]

```
GarbageCollected & GarbageCollected::operator= (  
    GarbageCollected && other )
```

Move Assignment.

Parameters

<i>The</i>	other GarbageCollected object.
------------	--

5.35.3.18 operator==() [1/8]

```
bool GarbageCollected::operator== (  
    const bool & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

5.35.3.19 operator==() [2/8]

```
bool GarbageCollected::operator== (
    const char *const & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

5.35.3.20 operator==() [3/8]

```
bool GarbageCollected::operator== (
    const Error & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

5.35.3.21 operator==() [4/8]

```
GarbageCollected GarbageCollected::operator== (
    const GarbageCollected & rhs ) const
```

Perform a == between two [GarbageCollected](#) values.

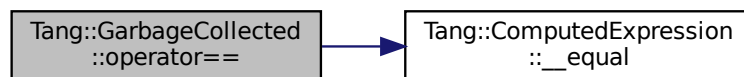
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:

**5.35.3.22 operator==() [5/8]**

```
bool GarbageCollected::operator==(  
    const std::nullptr_t & null ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

5.35.3.23 operator==() [6/8]

```
bool GarbageCollected::operator==(  
    const std::string & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

5.35.3.24 operator==() [7/8]

```
bool GarbageCollected::operator== (
    const Tang::float_t & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

5.35.3.25 operator==() [8/8]

```
bool GarbageCollected::operator== (
    const Tang::integer_t & val ) const
```

Compare the [GarbageCollected](#) tracked object with a supplied value.

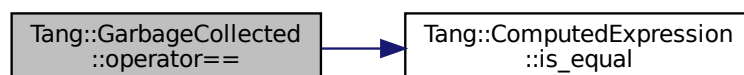
Parameters

<i>val</i>	The value to compare the tracked object against.
------------	--

Returns

True if they are equal, false otherwise.

Here is the call graph for this function:



5.35.3.26 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.

5.35.3.27 operator>=()

```
GarbageCollected GarbageCollected::operator>= (
    const GarbageCollected & rhs ) const
```

Perform a >= between two [GarbageCollected](#) values.

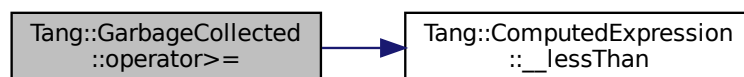
Parameters

<i>rhs</i>	The right hand side operand.
------------	------------------------------

Returns

The result of the operation.

Here is the call graph for this function:



5.35.4 Friends And Related Function Documentation

5.35.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & out,
    const GarbageCollected & gc ) [friend]
```

Add friendly output.

Parameters

<i>out</i>	The output stream.
<i>gc</i>	The GarbageCollected value.

Returns

The output stream.

The documentation for this class was generated from the following files:

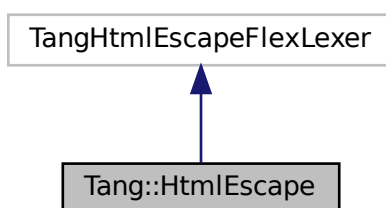
- [include/garbageCollected.hpp](#)
- [src/garbageCollected.cpp](#)

5.36 Tang::HtmlEscape Class Reference

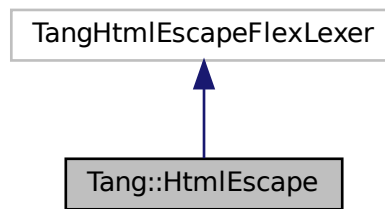
The Flex lexer class for the main Tang language.

```
#include <htmlEscape.hpp>
```

Inheritance diagram for Tang::HtmlEscape:



Collaboration diagram for Tang::HtmlEscape:



Public Member Functions

- [HtmlEscape](#) (std::istream &arg_yyin, std::ostream &arg_yyout)
The constructor for the Scanner.
- virtual std::string [get_next_token](#) ()
Extract the next token from the input string.

5.36.1 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.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 HtmlEscape()

```
Tang::HtmlEscape::HtmlEscape (  
    std::istream & arg_yyin,  
    std::ostream & arg_yyout ) [inline]
```

The constructor for the Scanner.

The design of the Flex lexer is to tokenize the contents of an input stream, and to write any error messages to an output stream. In our implementation, however, errors are returned differently, so the output stream is never used. 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)

5.36.3 Member Function Documentation

5.36.3.1 `get_next_token()`

```
virtual std::string Tang::HtmlEscape::get_next_token ( ) [virtual]
```

Extract the next token from the input string.

Returns

The next unescaped character.

The documentation for this class was generated from the following file:

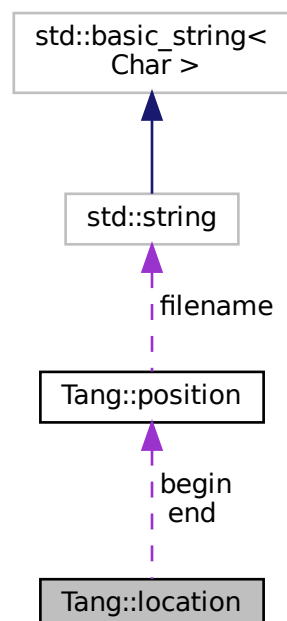
- [include/htmlEscape.hpp](#)

5.37 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.

5.37.1 Detailed Description

Two points in a source file.

The documentation for this class was generated from the following file:

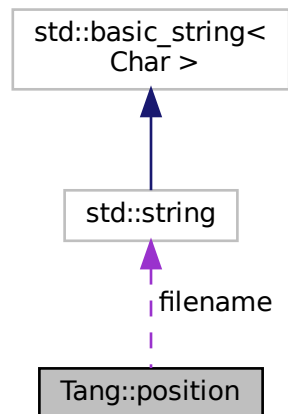
- build/generated/[location.hh](#)

5.38 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.

5.38.1 Detailed Description

A point in a source file.

The documentation for this class was generated from the following file:

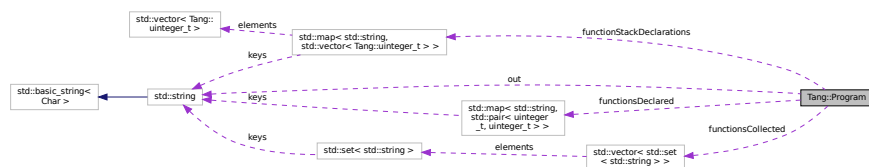
- [build/generated/location.hh](#)

5.39 Tang::Program Class Reference

Represents a compiled script or template that may be executed.

```
#include <program.hpp>
```

Collaboration diagram for Tang::Program:



Public Types

- enum [CodeType](#) { [Script](#) , [Template](#) }
Indicate the type of code that was supplied to the [Program](#).

Public Member Functions

- [Program](#) (std::string code, [CodeType](#) codeType)
Create a compiled program using the provided code.
- std::string [getCode](#) () const
Get the code that was provided when the [Program](#) was created.
- std::optional< const std::shared_ptr< [AstNode](#) > > [getAst](#) () const
Get the AST that was generated by the parser.
- std::string [dumpBytecode](#) () const
Get the Opcodes of the compiled program, formatted like Assembly.
- std::optional< const [GarbageCollected](#) > [getResult](#) () const
Get the result of the [Program](#) execution, if it exists.
- size_t [addBytecode](#) ([Tang::uinteger_t](#))
Add a [Tang::uinteger_t](#) to the Bytecode.
- const [Bytecode](#) & [getBytecode](#) ()
Get the Bytecode vector.
- [Program](#) & [execute](#) ()
Execute the program's Bytecode, and return the current [Program](#) object.
- bool [setJumpTarget](#) (size_t opcodeAddress, [Tang::uinteger_t](#) jumpTarget)
Set the target address of a Jump opcode.
- bool [setFunctionStackDeclaration](#) (size_t opcodeAddress, [uinteger_t](#) argc, [uinteger_t](#) targetPC)
Set the stack details of a function declaration.
- void [pushEnvironment](#) (const std::shared_ptr< [AstNode](#) > &ast)
Create a new compile/execute environment stack entry.
- void [popEnvironment](#) ()
Remove a compile/execute environment stack entry.
- void [addIdentifier](#) (const std::string &name, std::optional< size_t > [position](#)={})
Add an identifier to the environment.
- const std::map< std::string, size_t > & [getIdentifiers](#) () const
Get the identifier map of the current environment.
- void [addIdentifierAssigned](#) (const std::string &name)
Indicate that an identifier will be altered within the associated scope.
- const std::set< std::string > & [getIdentifiersAssigned](#) () const
Get the set of identifiers that will be assigned in the current scope.
- void [addString](#) (const std::string &name)
Add a string to the environment.
- const std::map< std::string, size_t > & [getStrings](#) () const
Get the string map of the current environment.
- void [pushBreakStack](#) ()
*Increase the *break* environment stack, so that we can handle nested break-supporting structures.*
- void [addBreak](#) (size_t location)
*Add the Bytecode location of a *break* statement, to be set when the final target is known at a later time.*
- void [popBreakStack](#) (size_t target)
*For all *continue* bytecode locations collected by [Tang::addContinue](#), set the target pc to target.*
- void [pushContinueStack](#) ()
*Increase the *continue* environment stack, so that we can handle nested continue-supporting structures.*
- void [addContinue](#) (size_t location)
*Add the Bytecode location of a *continue* statement, to be set when the final target is known at a later time.*
- void [popContinueStack](#) (size_t target)
*For all *continue* bytecode locations collected by [Tang::addContinue](#), set the target pc to target.*

Public Attributes

- `std::string out`
The output of the program, resulting from the program execution.
- `std::vector< std::set< std::string > > functionsCollected`
Names of the functions that are declared in a previous or the current scope.
- `std::map< std::string, std::pair< uinteger_t, uinteger_t > > functionsDeclared`
Key/value pair of the function declaration information.
- `std::map< std::string, std::vector< Tang::uinteger_t > > functionStackDeclarations`
For each function name, a list of Bytecode addresses that need to be replaced by a function definition.

5.39.1 Detailed Description

Represents a compiled script or template that may be executed.

5.39.2 Member Enumeration Documentation

5.39.2.1 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.

5.39.3 Constructor & Destructor Documentation

5.39.3.1 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> .

5.39.4 Member Function Documentation

5.39.4.1 addBreak()

```
void Program::addBreak (
    size_t location )
```

Add the Bytecode location of a `break` statement, to be set when the final target is known at a later time.

Parameters

<i>location</i>	The offset location of the <code>break</code> bytecode.
-----------------	---

5.39.4.2 addBytecode()

```
size_t Program::addBytecode (
    Tang::uinteger_t op )
```

Add a `Tang::uinteger_t` to the Bytecode.

Parameters

<i>op</i>	The value to add to the Bytecode.
-----------	-----------------------------------

Returns

The size of the bytecode structure.

5.39.4.3 addContinue()

```
void Program::addContinue (
    size_t location )
```

Add the Bytecode location of a `continue` statement, to be set when the final target is known at a later time.

Parameters

<i>location</i>	The offset location of the <code>continue</code> bytecode.
-----------------	--

5.39.4.4 addIdentifier()

```
void Program::addIdentifier (
    const std::string & name,
    std::optional< size_t > position = {} )
```

Add an identifier to the environment.

Parameters

<i>name</i>	The variable to add to the environment.
<i>position</i>	If provided, the desired position to place the identifier.

5.39.4.5 addIdentifierAssigned()

```
void Program::addIdentifierAssigned (
    const std::string & name )
```

Indicate that an identifier will be altered within the associated scope.

Parameters

<i>name</i>	The identifier name.
-------------	----------------------

5.39.4.6 addString()

```
void Program::addString (
    const std::string & name )
```

Add a string to the environment.

Parameters

<i>name</i>	The variable to add to the environment.
<i>position</i>	If provided, the desired position to place the identifier.

5.39.4.7 dumpBytecode()

```
string Program::dumpBytecode ( ) const
```

Get the Opcodes of the compiled program, formatted like Assembly.

Returns

A string containing the Opcode representation.

5.39.4.8 execute()

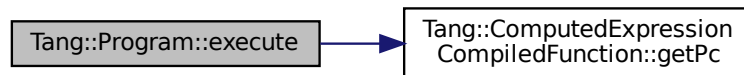
```
Program & Program::execute ( )
```

Execute the program's Bytecode, and return the current [Program](#) object.

Returns

The current [Program](#) object.

Here is the call graph for this function:

**5.39.4.9 getAst()**

```
optional< const shared_ptr< AstNode > > Program::getAst ( ) const
```

Get the AST that was generated by the parser.

The parser may have failed, so the return is an `optional<>` type. If the compilation failed, check `Program::error`.

Returns

A pointer to the AST, if it exists.

5.39.4.10 getBytecode()

```
const Bytecode & Program::getBytecode ( )
```

Get the Bytecode vector.

Returns

The Bytecode vector.

5.39.4.11 getCode()

```
string Program::getCode ( ) const
```

Get the code that was provided when the [Program](#) was created.

Returns

The source code from which the [Program](#) was created.

5.39.4.12 getIdentifiers()

```
const map< string, size_t > & Program::getIdentifiers ( ) const
```

Get the identifier map of the current environment.

Returns

A map of each identifier name to its stack position within the current environment.

5.39.4.13 getIdentifiersAssigned()

```
const set< string > & Program::getIdentifiersAssigned ( ) const
```

Get the set of identifiers that will be assigned in the current scope.

Returns

A set of identifier names that have been identified as the target of an assignment operator within the current scope.

5.39.4.14 getResult()

```
optional< const GarbageCollected > Program::getResult ( ) const
```

Get the result of the [Program](#) execution, if it exists.

Returns

The result of the [Program](#) execution, if it exists.

5.39.4.15 getStrings()

```
const map< string, size_t > & Program::getStrings ( ) const
```

Get the string map of the current environment.

Returns

A map of each identifier name to its stack position within the current environment.

5.39.4.16 popBreakStack()

```
void Program::popBreakStack (
    size_t target )
```

For all `continue` bytecode locations collected by `Tang::addContinue`, set the target pc to `target`.

Parameters

<i>target</i>	The target bytecode offset that the <code>continue</code> should jump to.
---------------	---

Here is the call graph for this function:



5.39.4.17 popContinueStack()

```
void Program::popContinueStack (
    size_t target )
```

For all `continue` bytecode locations collected by `Tang::addContinue`, set the target pc to `target`.

Parameters

<i>target</i>	The target bytecode offset that the <code>continue</code> should jump to.
---------------	---

Here is the call graph for this function:



5.39.4.18 pushEnvironment()

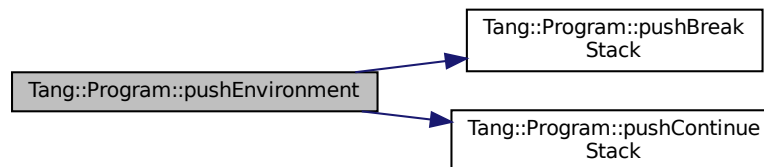
```
void Program::pushEnvironment (
    const std::shared_ptr< AstNode > & ast )
```

Create a new compile/execute environment stack entry.

Parameters

<code>ast</code>	The ast node from which this new environment will be formed.
------------------	--

Here is the call graph for this function:



5.39.4.19 setFunctionStackDeclaration()

```
bool Program::setFunctionStackDeclaration (
    size_t opcodeAddress,
    unsigned_t argc,
    unsigned_t targetPC )
```

Set the stack details of a function declaration.

Parameters

<i>opcodeAddress</i>	The location of the FUNCTION opcode.
<i>argc</i>	The argument count to set.
<i>targetPC</i>	The bytecode address of the start of the function.

5.39.4.20 setJumpTarget()

```
bool Program::setJumpTarget (
    size_t opcodeAddress,
    Tang::uinteger_t jumpTarget )
```

Set the target address of a Jump opcode.

Parameters

<i>opcodeAddress</i>	The location of the jump statement.
<i>jumpTarget</i>	The address to jump to.

Returns

Whether or not the jumpTarget was set.

5.39.5 Member Data Documentation**5.39.5.1 functionsDeclared**

```
std::map<std::string, std::pair<uinteger_t, uinteger_t> > Tang::Program::functionsDeclared
```

Key/value pair of the function declaration information.

The key is the name of the function. The value is a pair of the *argc* value and the *targetPC* value.

The documentation for this class was generated from the following files:

- [include/program.hpp](#)
- [src/program-dumpBytecode.cpp](#)
- [src/program-execute.cpp](#)
- [src/program.cpp](#)

5.40 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.

5.40.1 Detailed Description

```
template<class T>
class Tang::SingletonObjectPool< T >
```

A thread-safe, singleton object pool of the designated type.

5.40.2 Member Function Documentation

5.40.2.1 [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.

5.40.2.2 [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.

5.40.2.3 [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](#)

5.41 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](#).

5.41.1 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).

5.41.2 Constructor & Destructor Documentation

5.41.2.1 TangBase()

```
TangBase::TangBase ( )
```

The constructor.

Isn't it glorious.

5.41.3 Member Function Documentation

5.41.3.1 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:

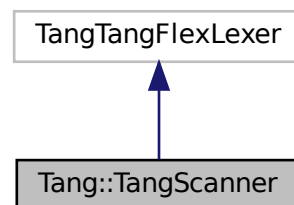
- [include/tangBase.hpp](#)
- [src/tangBase.cpp](#)

5.42 Tang::TangScanner Class Reference

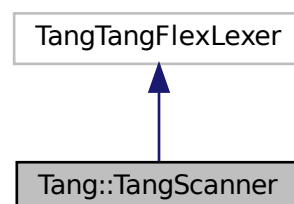
The Flex lexer class for the main Tang language.

```
#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.

5.42.1 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.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 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)

5.42.3 Member Function Documentation

5.42.3.1 get_next_token()

```
virtual Tang::TangParser::symbol_type Tang::TangScanner::get_next_token ( ) [virtual]
```

A pass-through function that we supply so that we can provide a Bison 3 token return type instead of the `int` that is returned by the default class configuration.

Returns

A Bison 3 token representing the lexeme that was recognized.

The documentation for this class was generated from the following file:

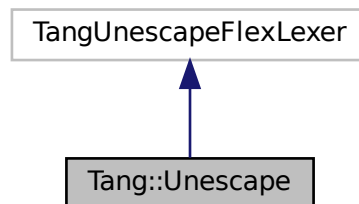
- include/[tangScanner.hpp](#)

5.43 Tang::Unescape Class Reference

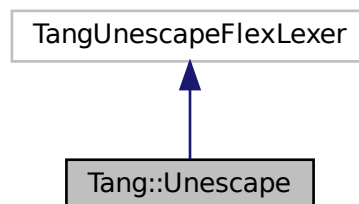
The Flex lexer class for the main Tang language.

```
#include <unescape.hpp>
```

Inheritance diagram for Tang::Unescape:



Collaboration diagram for Tang::Unescape:



Public Member Functions

- [Unescape](#) (std::istream &arg_yyin, std::ostream &arg_yyout)
The constructor for the Scanner.
- virtual std::string [get_next_token](#) ()
Extract the next token from the input string.

5.43.1 Detailed Description

The Flex lexer class for the main Tang language.

Flex requires that our lexer class inherit from `yyFlexLexer`, an "intermediate" class whose real name is "`TangTangFlexLexer`". We are subclassing it so that we can override the return type of `get_next_token()`, for compatibility with Bison 3 tokens.

5.43.2 Constructor & Destructor Documentation

5.43.2.1 Unescape()

```
Tang::Unescape::Unescape (
    std::istream & arg_yyin,
    std::ostream & arg_yyout ) [inline]
```

The constructor for the Scanner.

The design of the Flex lexer is to tokenize the contents of an input stream, and to write any error messages to an output stream. In our implementation, however, errors are returned differently, so the output stream is never used. 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)

5.43.3 Member Function Documentation

5.43.3.1 get_next_token()

```
virtual std::string Tang::Unescape::get_next_token ( ) [virtual]
```

Extract the next token from the input string.

Returns

The next unescaped character.

The documentation for this class was generated from the following file:

- [include/unescape.hpp](#)

5.44 Tang::UnicodeString Class Reference

Represents a UTF-8 encoded string that is Unicode-aware.

```
#include <unicodeString.hpp>
```

Public Member Functions

- [UnicodeString](#) (const std::string &src)
Construct a [Tang::UnicodeString](#) object, which acts as the interface to the ICU library.
- std::string [substr](#) (size_t [position](#), size_t [length](#)) const
Return a Unicode grapheme-aware substring.
- bool [operator==](#) (const [UnicodeString](#) &rhs) const
Compare two UnicodeStrings.
- bool [operator<](#) (const [UnicodeString](#) &rhs) const
Compare two UnicodeStrings.
- [UnicodeString operator+](#) (const [UnicodeString](#) &rhs) const
Create a new [UnicodeString](#) that is the concatenation of two UnicodeStrings.
- [operator std::string](#) () const
Cast the current [UnicodeString](#) object to a std::string, UTF-8 encoded.
- size_t [length](#) () const
Return the length of the [UnicodeString](#) in graphemes.
- size_t [bytesLength](#) () const
Return the length of the [UnicodeString](#) in bytes.

5.44.1 Detailed Description

Represents a UTF-8 encoded string that is Unicode-aware.

This class serves as the interface between the Tang language and the ICU library.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 UnicodeString()

```
UnicodeString::UnicodeString (  
    const std::string & src )
```

Construct a [Tang::UnicodeString](#) object, which acts as the interface to the ICU library.

Parameters

<i>src</i>	A UTF-8 encoded string.
------------	-------------------------

5.44.3 Member Function Documentation

5.44.3.1 `bytesLength()`

```
size_t UnicodeString::bytesLength ( ) const
```

Return the length of the [UnicodeString](#) in bytes.

Note: this is *not* the number of codepoints or graphemes, but is the actual number of bytes in memory.

Returns

Returns the length of the [UnicodeString](#) in bytes.

5.44.3.2 `length()`

```
size_t UnicodeString::length ( ) const
```

Return the length of the [UnicodeString](#) in graphemes.

Note: this is *not* the number of bytes, chars, or codepoints, but is the length in graphemes, as defined by ICU.

Returns

Returns the length of the [UnicodeString](#) in graphemes.

5.44.3.3 `operator std::string()`

```
UnicodeString::operator std::string ( ) const
```

Cast the current [UnicodeString](#) object to a `std::string`, UTF-8 encoded.

Returns

Returns the `std::string` version of the [UnicodeString](#).

5.44.3.4 `operator+()`

```
UnicodeString UnicodeString::operator+ (
    const UnicodeString & rhs ) const
```

Create a new [UnicodeString](#) that is the concatenation of two `UnicodeStrings`.

Parameters

<i>rhs</i>	The string to append to the current object string.
------------	--

Returns

Returns the result of the concatenation.

5.44.3.5 operator<()

```
bool UnicodeString::operator< (
    const UnicodeString & rhs ) const
```

Compare two UnicodeStrings.

Parameters

<i>rhs</i>	The string to compare against.
------------	--------------------------------

Returns

Returns true if the rhs string is greater than or equal to the object string.

5.44.3.6 operator==()

```
bool UnicodeString::operator== (
    const UnicodeString & rhs ) const
```

Compare two UnicodeStrings.

Parameters

<i>rhs</i>	The string to compare against.
------------	--------------------------------

Returns

Returns true if the two strings are equal.

5.44.3.7 substr()

```
std::string UnicodeString::substr (
    size_t position,
    size_t length ) const
```

Return a Unicode grapheme-aware substring.

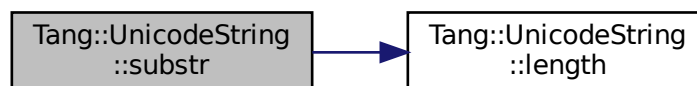
Parameters

<i>position</i>	The 0-based position of the first grapheme.
<i>length</i>	The maximum number of graphemes to return.

Returns

The requested substring.

Here is the call graph for this function:



The documentation for this class was generated from the following files:

- [include/unicodeString.hpp](#)
- [src/unicodeString.cpp](#)

Chapter 6

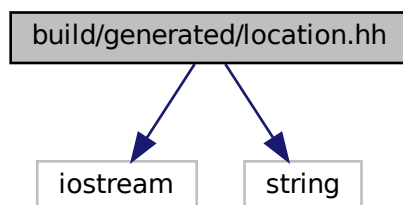
File Documentation

6.1 build/generated/location.hh File Reference

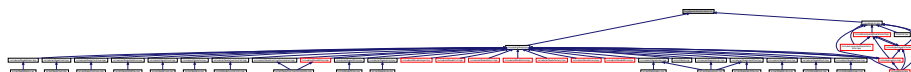
Define the Tang ::location class.

```
#include <iostream>
#include <string>
```

Include dependency graph for location.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class `Tang::position`
A point in a source file.
- class `Tang::location`
Two points in a source file.

Macros

- `#define YY_NULLPTR ((void*)0)`

Functions

- position & [Tang::operator+=](#) (position &res, position::counter_type width)
Add width columns, in place.
- position [Tang::operator+](#) (position res, position::counter_type width)
Add width columns.
- position & [Tang::operator-=](#) (position &res, position::counter_type width)
Subtract width columns, in place.
- position [Tang::operator-](#) (position res, position::counter_type width)
Subtract width columns.
- template<typename YYChar >
std::basic_ostream< YYChar > & [Tang::operator<<](#) (std::basic_ostream< YYChar > &ostr, const position &pos)
Intercept output stream redirection.
- location & [Tang::operator+=](#) (location &res, const location &end)
Join two locations, in place.
- location [Tang::operator+](#) (location res, const location &end)
Join two locations.
- location & [Tang::operator+=](#) (location &res, location::counter_type width)
Add width columns to the end position, in place.
- location [Tang::operator+](#) (location res, location::counter_type width)
Add width columns to the end position.
- location & [Tang::operator-=](#) (location &res, location::counter_type width)
Subtract width columns to the end position, in place.
- location [Tang::operator-](#) (location res, location::counter_type width)
Subtract width columns to the end position.
- template<typename YYChar >
std::basic_ostream< YYChar > & [Tang::operator<<](#) (std::basic_ostream< YYChar > &ostr, const location &loc)
Intercept output stream redirection.

6.1.1 Detailed Description

Define the Tang ::location class.

6.1.2 Function Documentation

6.1.2.1 operator<<() [1/2]

```
template<typename YYChar >
std::basic_ostream<YYChar>& Tang::operator<< (
    std::basic_ostream< YYChar > & ostr,
    const location & loc )
```

Intercept output stream redirection.

Parameters

<i>ostr</i>	the destination output stream
<i>loc</i>	a reference to the location to redirect

Avoid duplicate information.

6.1.2.2 operator<<() [2/2]

```
template<typename YYChar >
std::basic_ostream<YYChar>& Tang::operator<< (
    std::basic_ostream< YYChar > & ostr,
    const position & pos )
```

Intercept output stream redirection.

Parameters

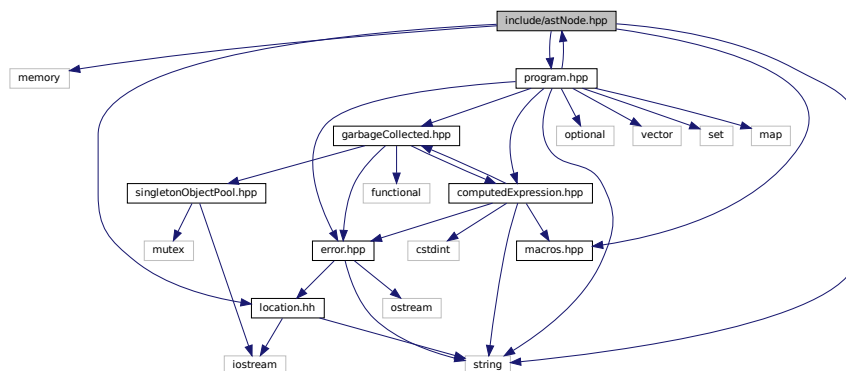
<i>ostr</i>	the destination output stream
<i>pos</i>	a reference to the position to redirect

6.2 include/astNode.hpp File Reference

Declare the [Tang::AstNode](#) base class.

```
#include <memory>
#include <string>
#include "location.hh"
#include "macros.hpp"
#include "program.hpp"
```

Include dependency graph for astNode.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNode](#)
Base class for representing nodes of an Abstract Syntax Tree (AST).

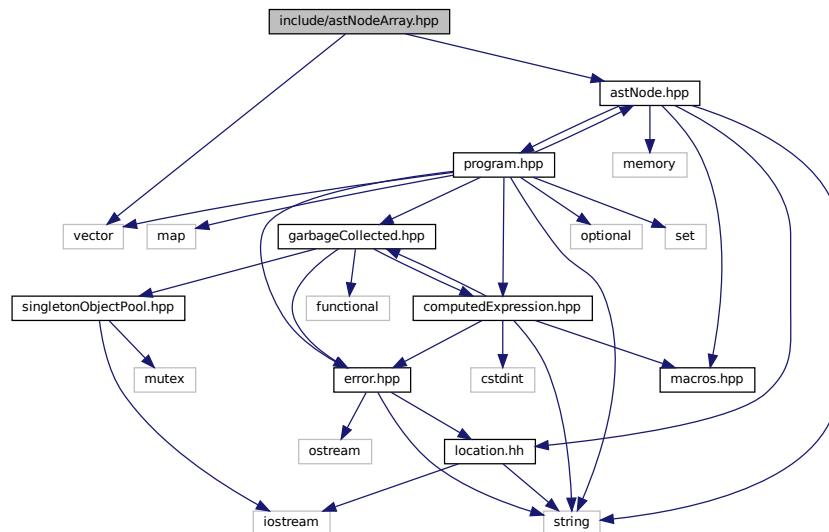
6.2.1 Detailed Description

Declare the [Tang::AstNode](#) base class.

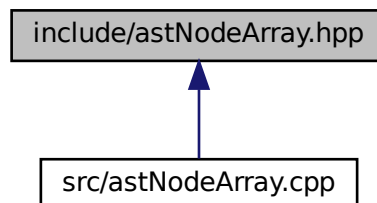
6.3 include/astNodeArray.hpp File Reference

Declare the [Tang::AstNodeArray](#) class.

```
#include <vector>
#include "astNode.hpp"
Include dependency graph for astNodeArray.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeArray](#)
An *AstNode* that represents an array literal.

6.3.1 Detailed Description

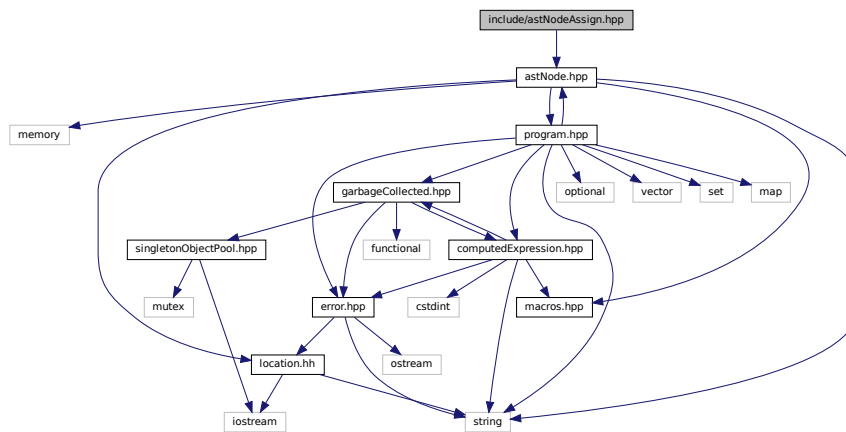
Declare the [Tang::AstNodeArray](#) class.

6.4 include/astNodeAssign.hpp File Reference

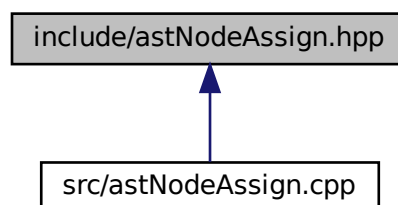
Declare the [Tang::AstNodeAssign](#) class.

```
#include "astNode.hpp"
```

Include dependency graph for astNodeAssign.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeAssign](#)
An [AstNode](#) that represents a binary expression.

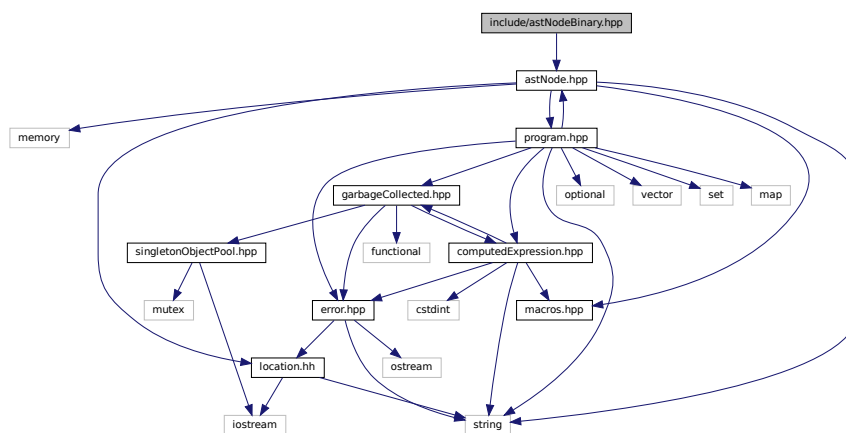
6.4.1 Detailed Description

Declare the [Tang::AstNodeAssign](#) class.

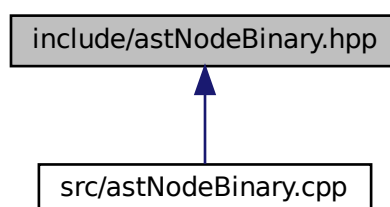
6.5 include/astNodeBinary.hpp File Reference

Declare the [Tang::AstNodeBinary](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeBinary.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeBinary](#)
An [AstNode](#) that represents a binary expression.

6.5.1 Detailed Description

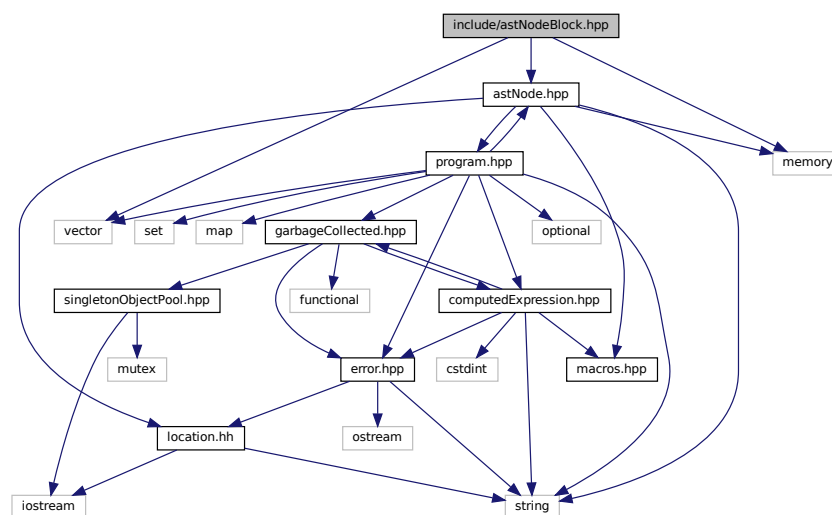
Declare the [Tang::AstNodeBinary](#) class.

6.6 include/astNodeBlock.hpp File Reference

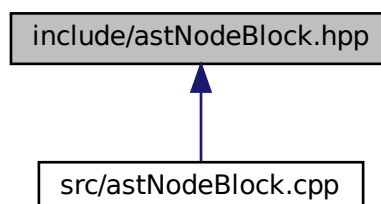
Declare the [Tang::AstNodeBlock](#) class.

```
#include <vector>
#include <memory>
#include "astNode.hpp"
```

Include dependency graph for astNodeBlock.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeBlock](#)
An *AstNode* that represents a code block.

6.6.1 Detailed Description

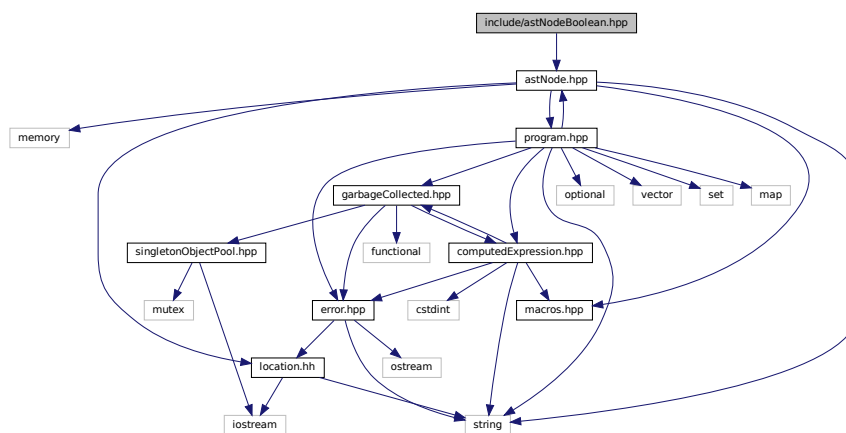
Declare the [Tang::AstNodeBlock](#) class.

6.7 include/astNodeBoolean.hpp File Reference

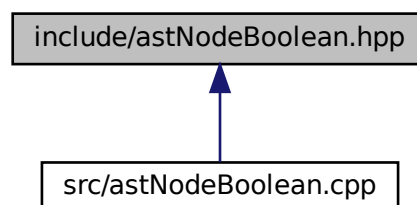
Declare the [Tang::AstNodeBoolean](#) class.

```
#include "astNode.hpp"
```

Include dependency graph for astNodeBoolean.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeBoolean](#)
An *AstNode* that represents a boolean literal.

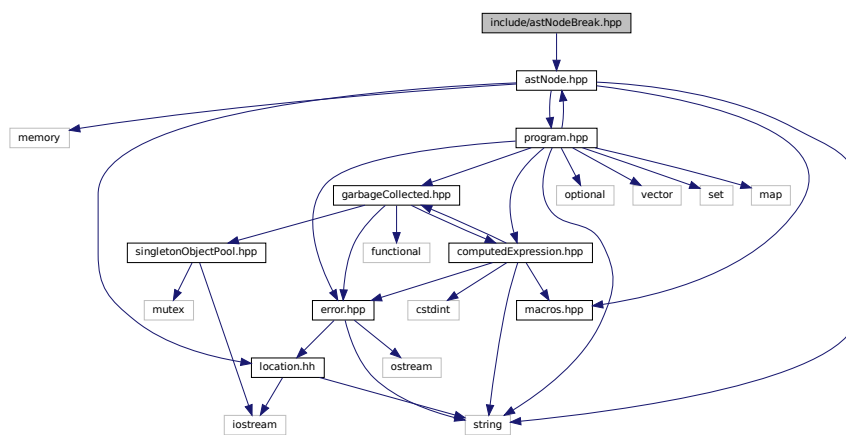
6.7.1 Detailed Description

Declare the [Tang::AstNodeBoolean](#) class.

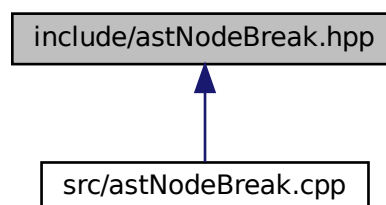
6.8 include/astNodeBreak.hpp File Reference

Declare the [Tang::AstNodeBreak](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeBreak.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeBreak](#)
An *AstNode* that represents a *break* statement.

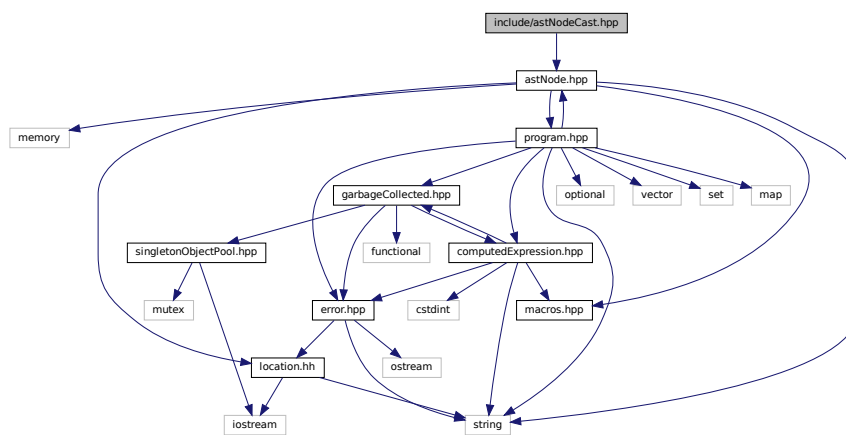
6.8.1 Detailed Description

Declare the [Tang::AstNodeBreak](#) class.

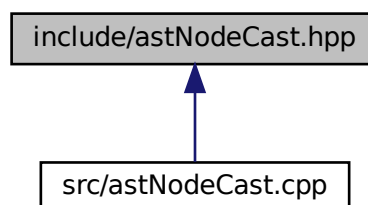
6.9 include/astNodeCast.hpp File Reference

Declare the [Tang::AstNodeCast](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeCast.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeCast](#)
An *AstNode* that represents a typecast of an expression.

6.9.1 Detailed Description

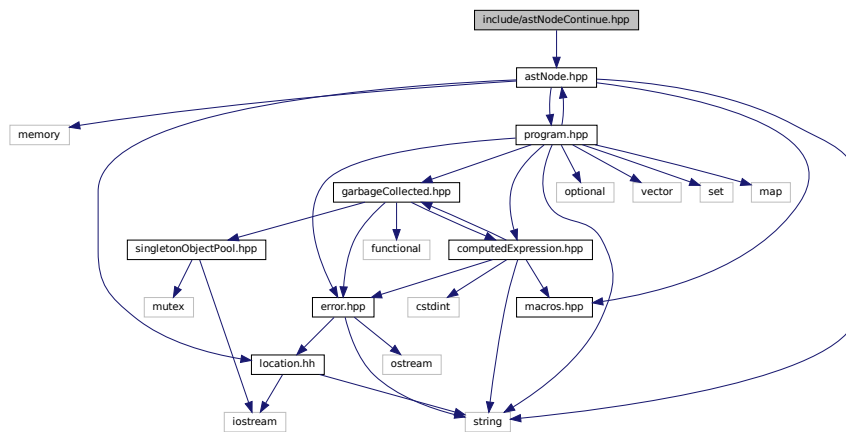
Declare the [Tang::AstNodeCast](#) class.

6.10 include/astNodeContinue.hpp File Reference

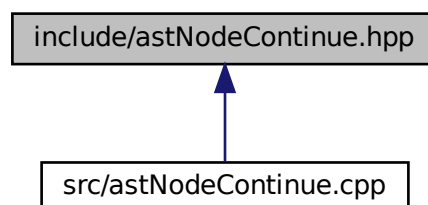
Declare the [Tang::AstNodeContinue](#) class.

```
#include "astNode.hpp"
```

Include dependency graph for astNodeContinue.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeContinue](#)

An [AstNode](#) that represents a *continue* statement.

6.10.1 Detailed Description

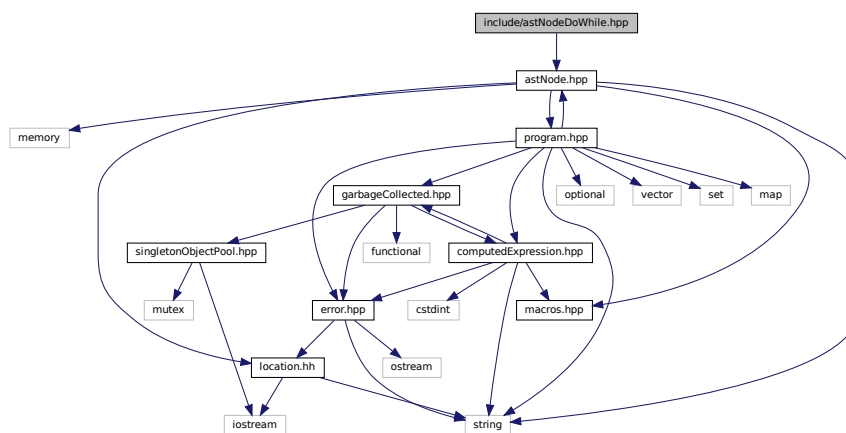
Declare the [Tang::AstNodeContinue](#) class.

6.11 include/astNodeDoWhile.hpp File Reference

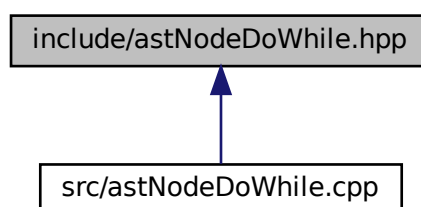
Declare the [Tang::AstNodeDoWhile](#) class.

```
#include "astNode.hpp"
```

Include dependency graph for astNodeDoWhile.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeDoWhile](#)
An *AstNode* that represents a do..while statement.

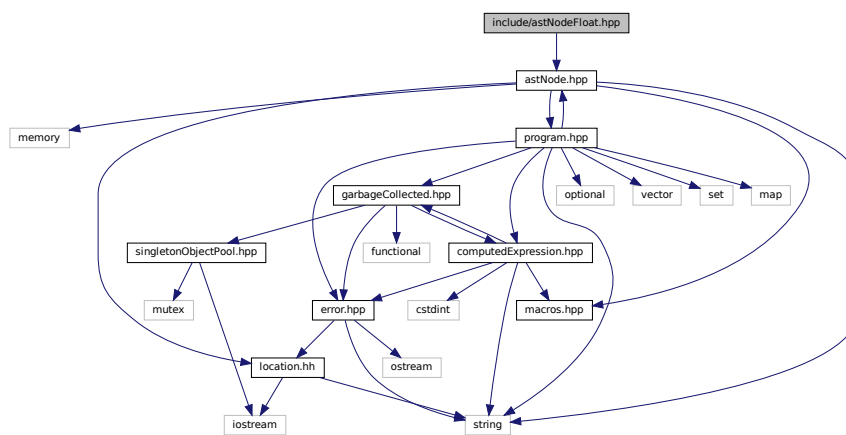
6.11.1 Detailed Description

Declare the [Tang::AstNodeDoWhile](#) class.

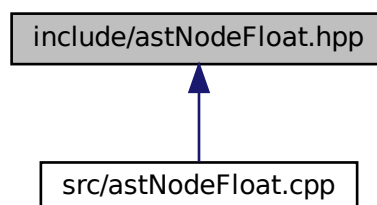
6.12 include/astNodeFloat.hpp File Reference

Declare the [Tang::AstNodeFloat](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeFloat.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeFloat](#)
An [AstNode](#) that represents an float literal.

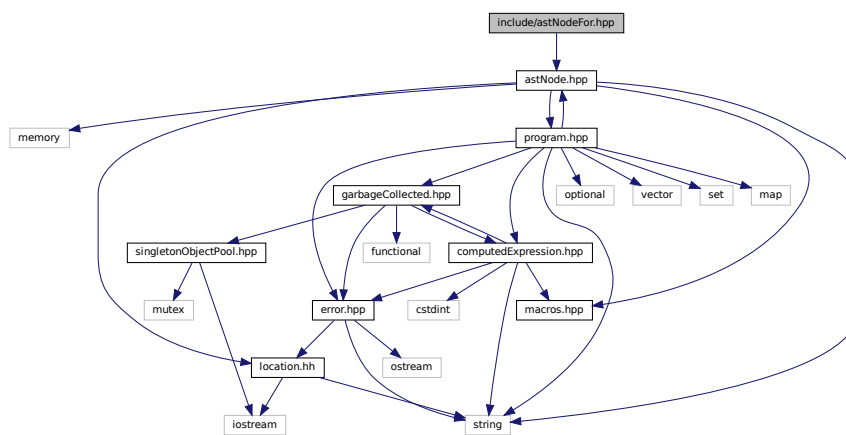
6.12.1 Detailed Description

Declare the [Tang::AstNodeFloat](#) class.

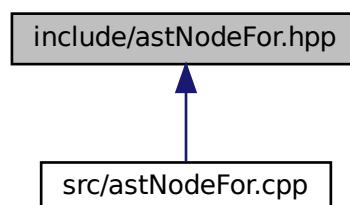
6.13 include/astNodeFor.hpp File Reference

Declare the [Tang::AstNodeFor](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeFor.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeFor](#)
An [AstNode](#) that represents an if() statement.

6.13.1 Detailed Description

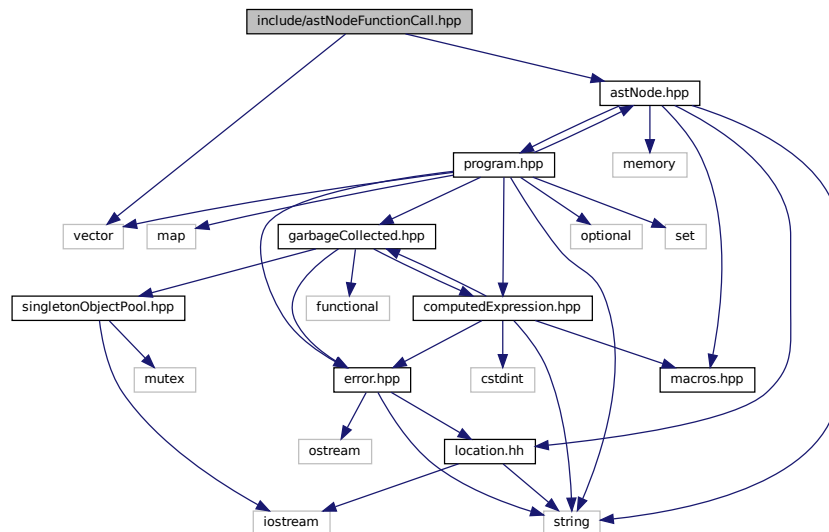
Declare the [Tang::AstNodeFor](#) class.

6.14 include/astNodeFunctionCall.hpp File Reference

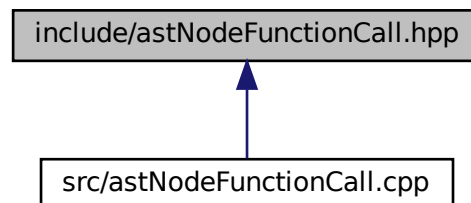
Declare the [Tang::AstNodeFunctionCall](#) class.

```
#include <vector>
#include "astNode.hpp"
```

Include dependency graph for astNodeFunctionCall.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeFunctionCall](#)
An [AstNode](#) that represents a function call.

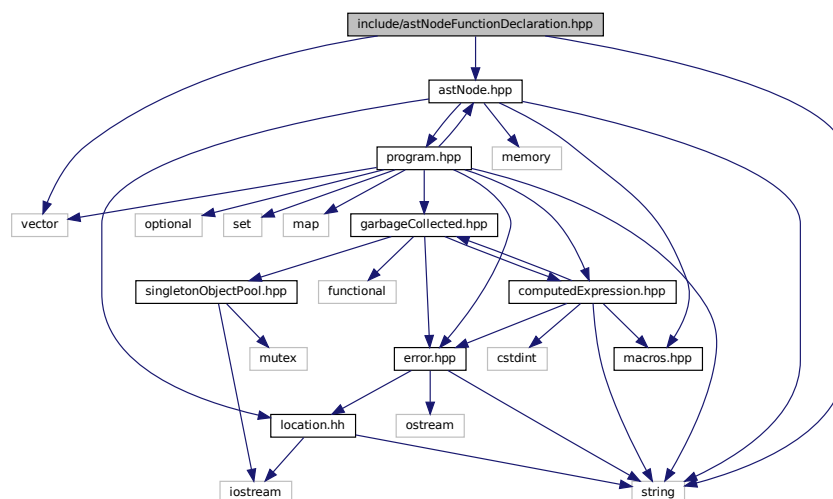
6.14.1 Detailed Description

Declare the [Tang::AstNodeFunctionCall](#) class.

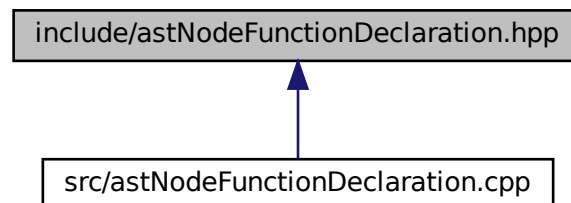
6.15 include/astNodeFunctionDeclaration.hpp File Reference

Declare the [Tang::AstNodeFunctionDeclaration](#) class.

```
#include <string>
#include <vector>
#include "astNode.hpp"
Include dependency graph for astNodeFunctionDeclaration.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeFunctionDeclaration](#)
An [AstNode](#) that represents a function declaration.

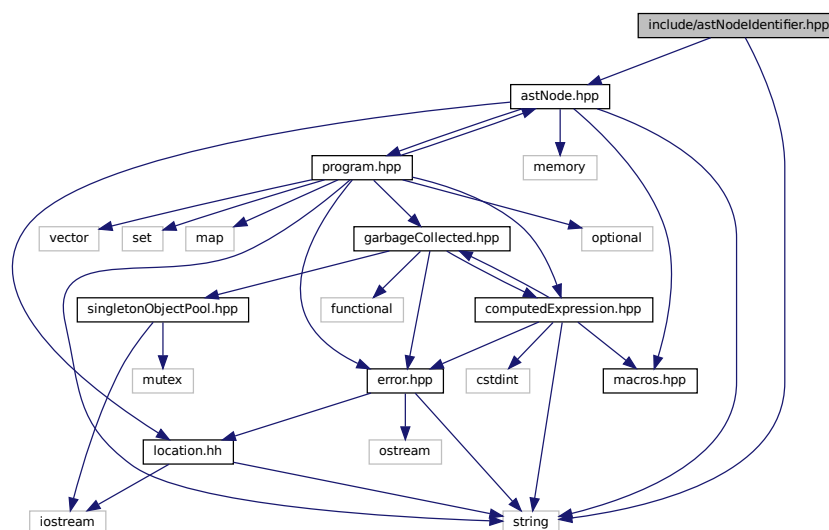
6.15.1 Detailed Description

Declare the [Tang::AstNodeFunctionDeclaration](#) class.

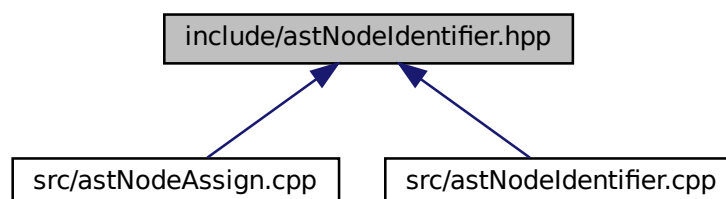
6.16 include/astNodeIdentifier.hpp File Reference

Declare the [Tang::AstNodeIdentifier](#) class.

```
#include <string>
#include "astNode.hpp"
Include dependency graph for astNodeIdentifier.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeIdentifier](#)
An [AstNode](#) that represents an identifier.

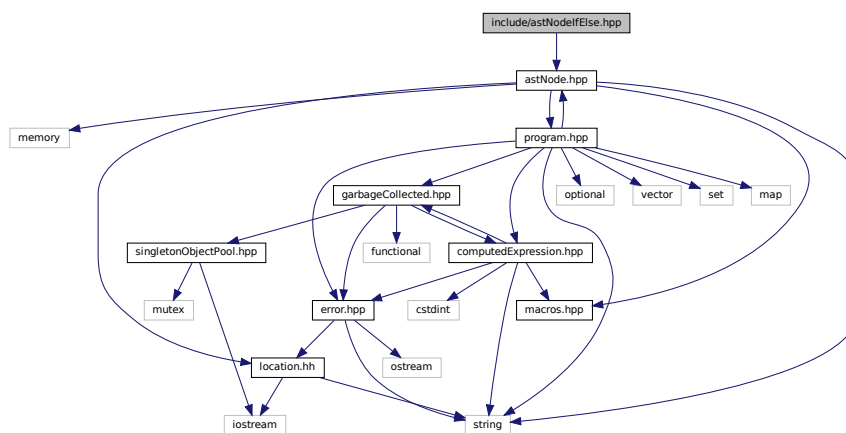
6.16.1 Detailed Description

Declare the [Tang::AstNodeIdentifier](#) class.

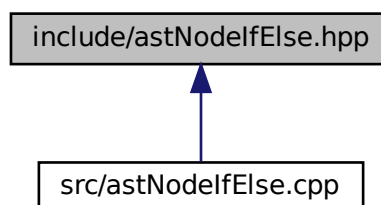
6.17 include/astNodeIfElse.hpp File Reference

Declare the [Tang::AstNodeIfElse](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeIfElse.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeIfElse](#)
An [AstNode](#) that represents an if..else statement.

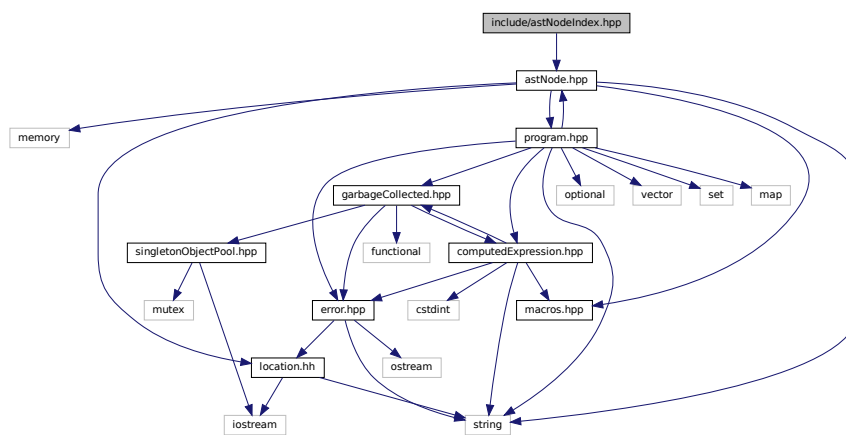
6.17.1 Detailed Description

Declare the [Tang::AstNodeIfElse](#) class.

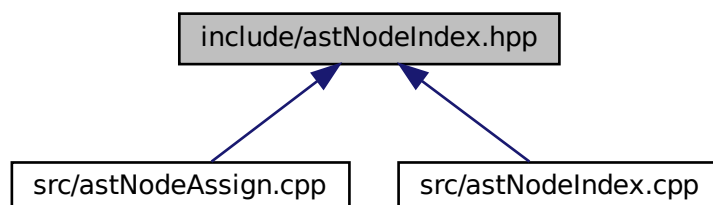
6.18 include/astNodeIndex.hpp File Reference

Declare the [Tang::AstNodeIndex](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeIndex.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeIndex](#)

An [AstNode](#) that represents an index into a collection.

6.18.1 Detailed Description

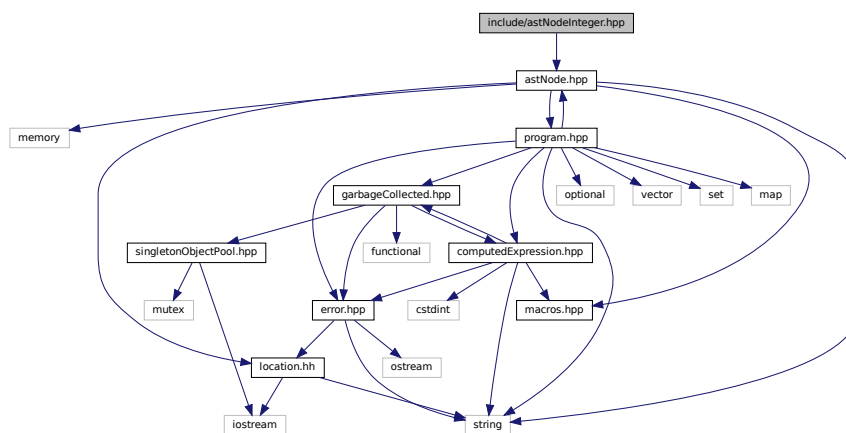
Declare the [Tang::AstNodeIndex](#) class.

6.19 include/astNodeInteger.hpp File Reference

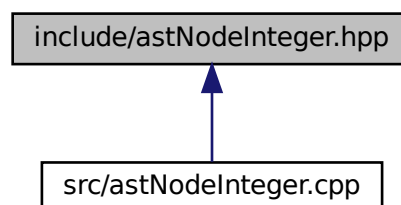
Declare the [Tang::AstNodeInteger](#) class.

```
#include "astNode.hpp"
```

Include dependency graph for astNodeInteger.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeInteger](#)
An *AstNode* that represents an integer literal.

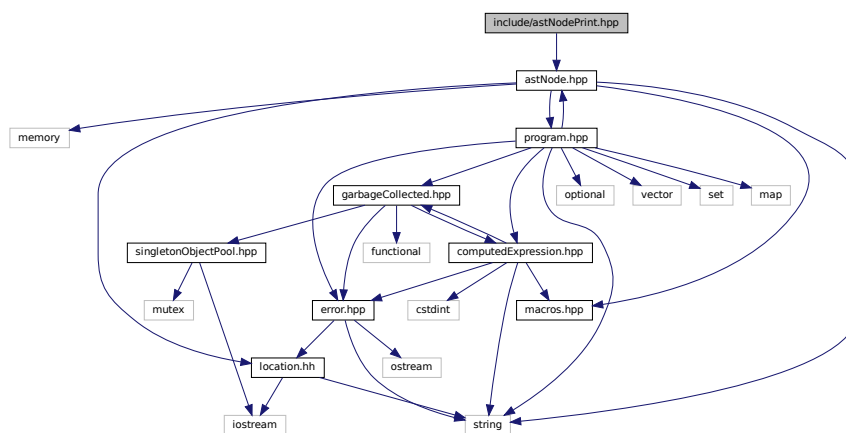
6.19.1 Detailed Description

Declare the [Tang::AstNodeInteger](#) class.

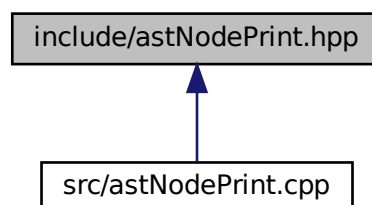
6.20 include/astNodePrint.hpp File Reference

Declare the [Tang::AstNodePrint](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodePrint.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodePrint](#)
An [AstNode](#) that represents a print typeoperation.

6.20.1 Detailed Description

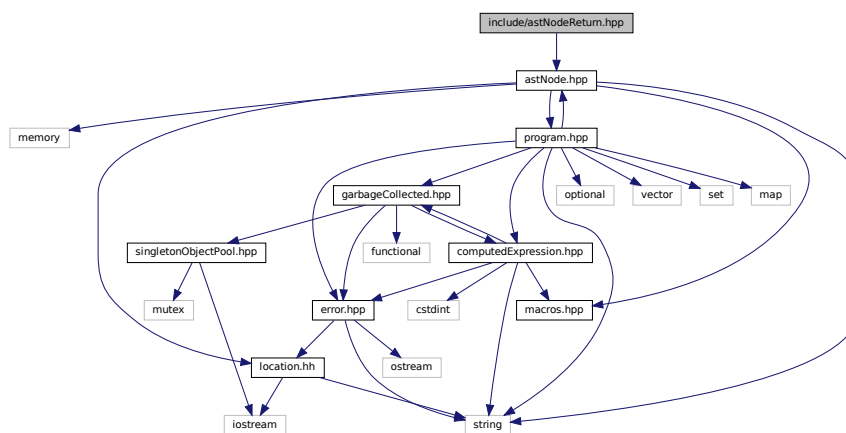
Declare the [Tang::AstNodePrint](#) class.

6.21 include/astNodeReturn.hpp File Reference

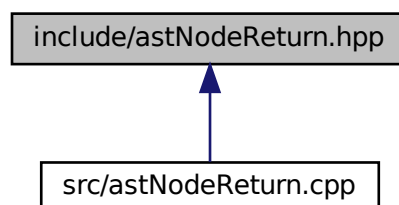
Declare the [Tang::AstNodeReturn](#) class.

```
#include "astNode.hpp"
```

Include dependency graph for astNodeReturn.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class `Tang::AstNodeReturn`

An *AstNode* that represents a *return* statement.

6.21.1 Detailed Description

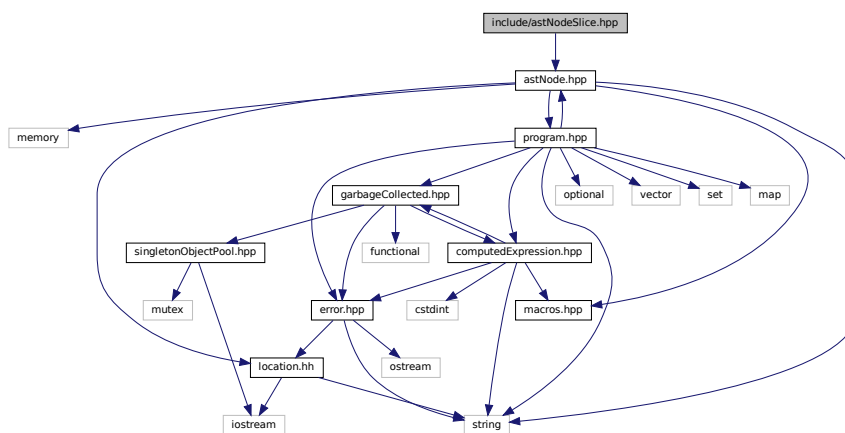
Declare the `Tang::AstNodeReturn` class.

6.22 include/astNodeSlice.hpp File Reference

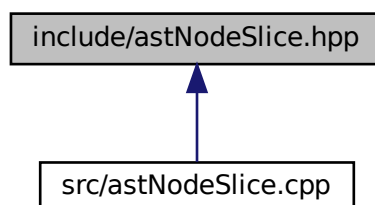
Declare the `Tang::AstNodeSlice` class.

```
#include "astNode.hpp"
```

Include dependency graph for astNodeSlice.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeSlice](#)
An *AstNode* that represents a ternary expression.

6.22.1 Detailed Description

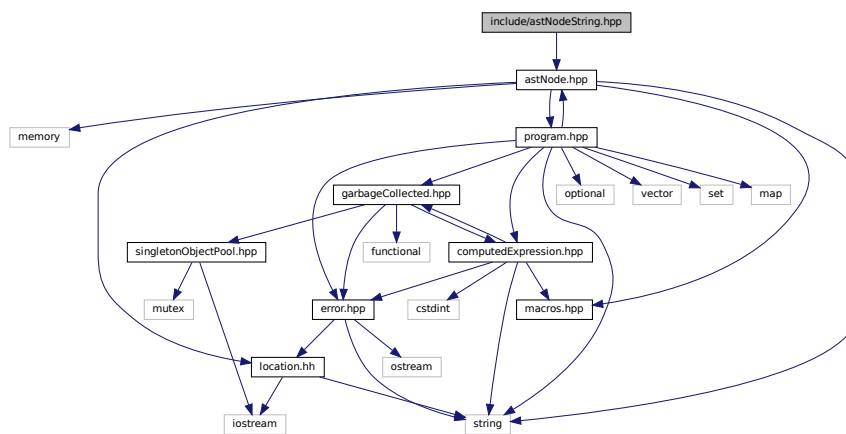
Declare the [Tang::AstNodeSlice](#) class.

6.23 include/astNodeString.hpp File Reference

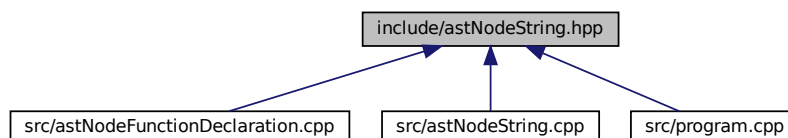
Declare the [Tang::AstNodeString](#) class.

```
#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.

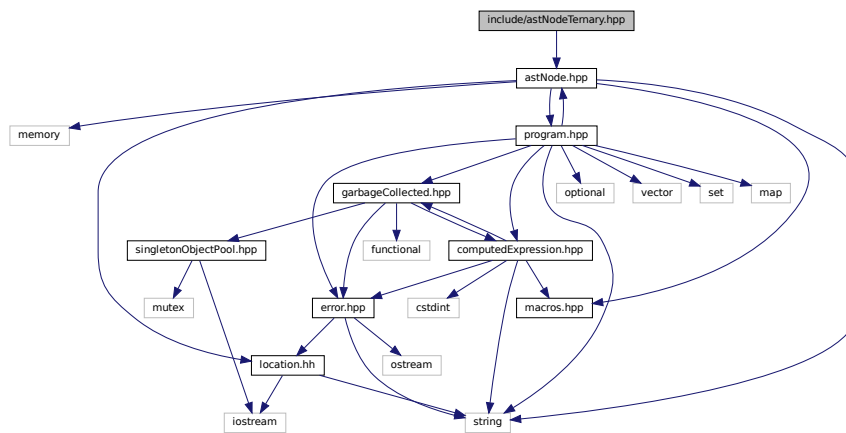
6.23.1 Detailed Description

Declare the [Tang::AstNodeString](#) class.

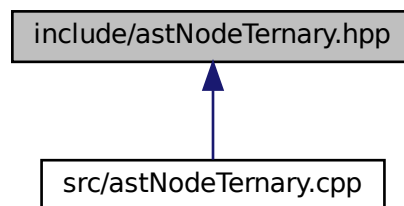
6.24 include/astNodeTernary.hpp File Reference

Declare the [Tang::AstNodeTernary](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeTernary.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeTernary](#)
An [AstNode](#) that represents a ternary expression.

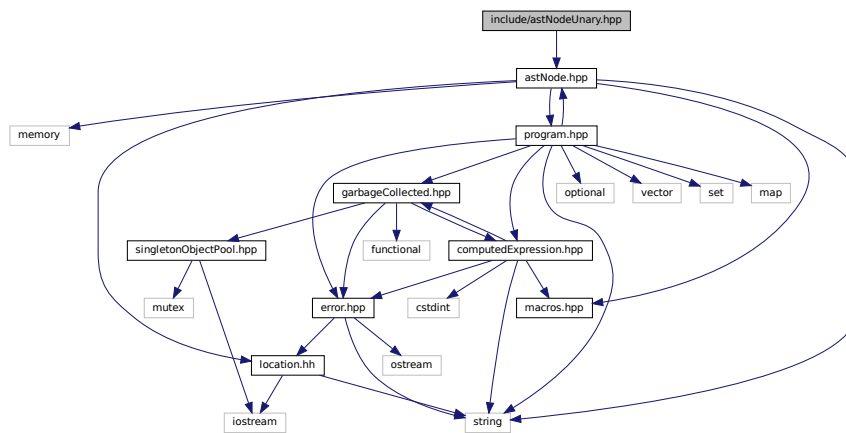
6.24.1 Detailed Description

Declare the [Tang::AstNodeTernary](#) class.

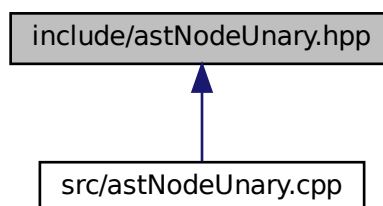
6.25 include/astNodeUnary.hpp File Reference

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.

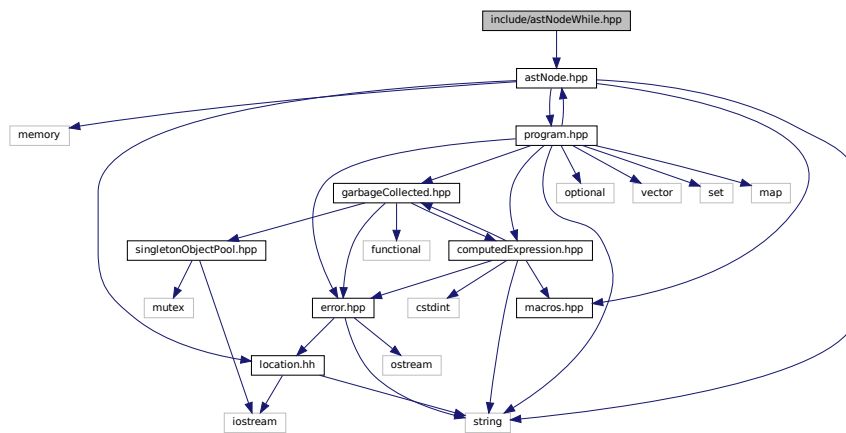
6.25.1 Detailed Description

Declare the [Tang::AstNodeUnary](#) class.

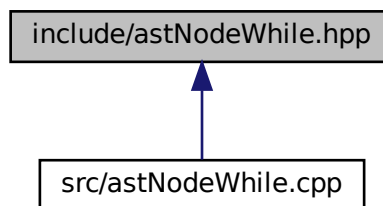
6.26 include/astNodeWhile.hpp File Reference

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.

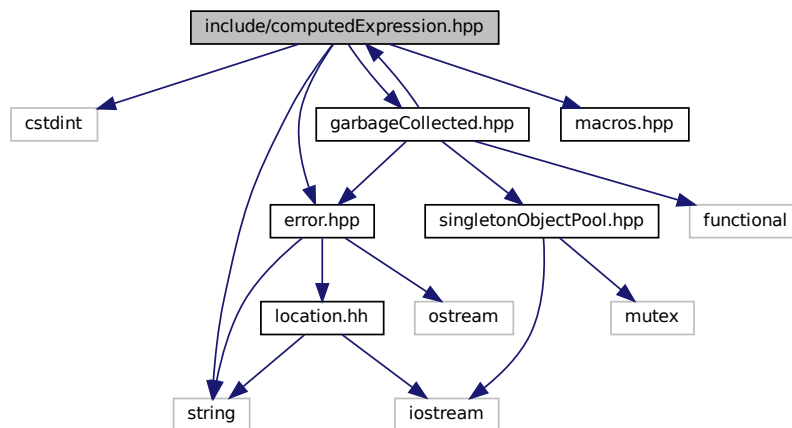
6.26.1 Detailed Description

Declare the [Tang::AstNodeWhile](#) class.

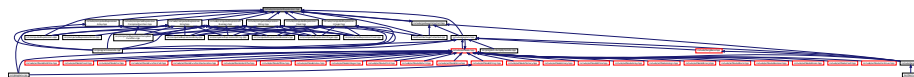
6.27 include/computedExpression.hpp File Reference

Declare the [Tang::ComputedExpression](#) base class.

```
#include <cstdint>
#include <string>
#include "macros.hpp"
#include "garbageCollected.hpp"
#include "error.hpp"
#include dependency graph for computedExpression.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpression](#)
Represents the result of a computation that has been executed.

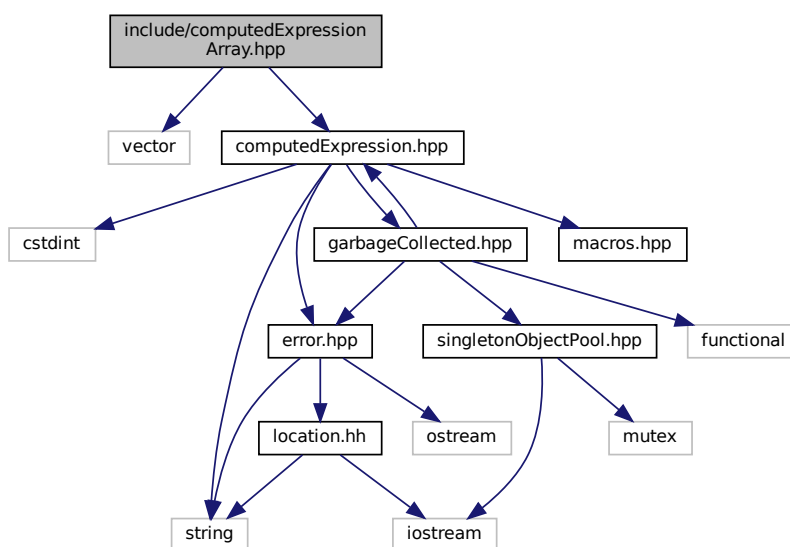
6.27.1 Detailed Description

Declare the [Tang::ComputedExpression](#) base class.

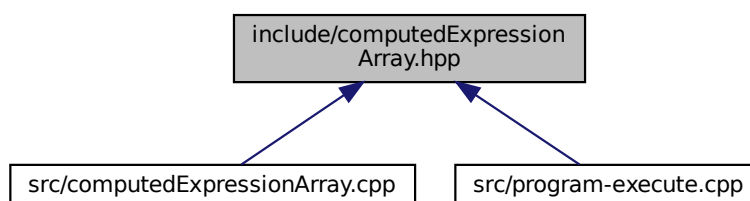
6.28 include/computedExpressionArray.hpp File Reference

Declare the `Tang::ComputedExpressionArray` class.

```
#include <vector>
#include "computedExpression.hpp"
Include dependency graph for computedExpressionArray.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class `Tang::ComputedExpressionArray`
Represents an Array that is the result of a computation.

6.28.1 Detailed Description

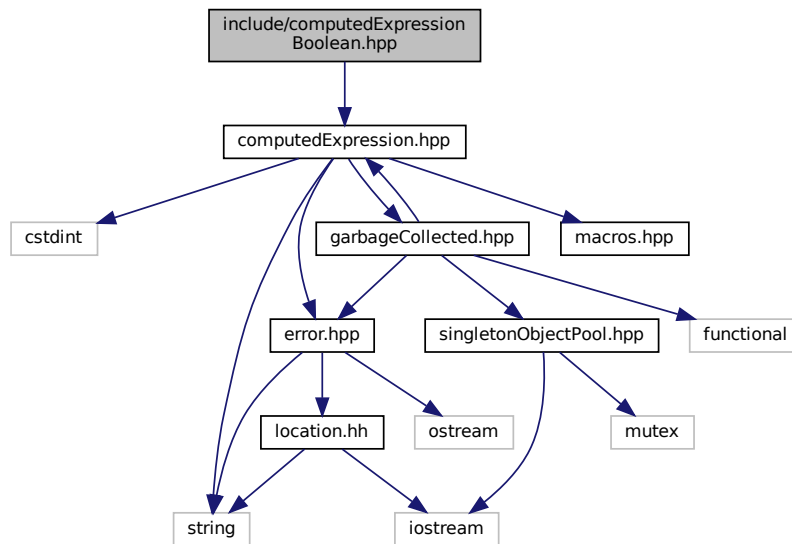
Declare the `Tang::ComputedExpressionArray` class.

6.29 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.

6.29.1 Detailed Description

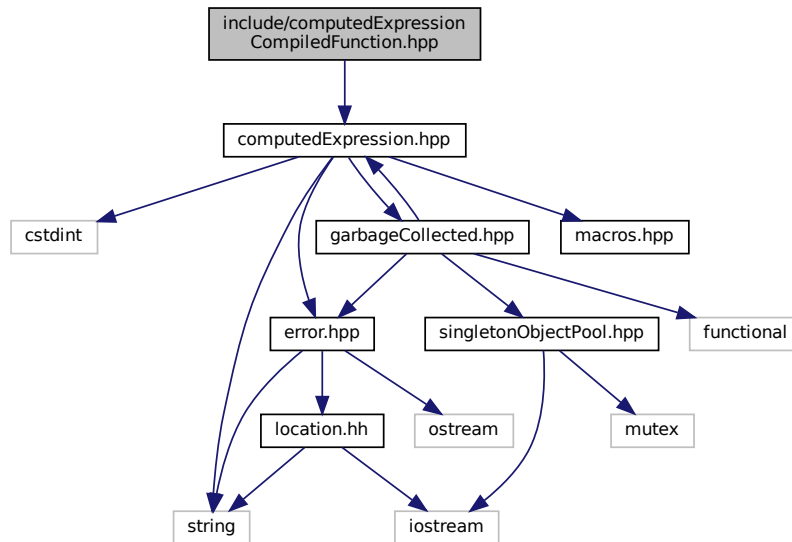
Declare the [Tang::ComputedExpressionBoolean](#) class.

6.30 include/computedExpressionCompiledFunction.hpp File Reference

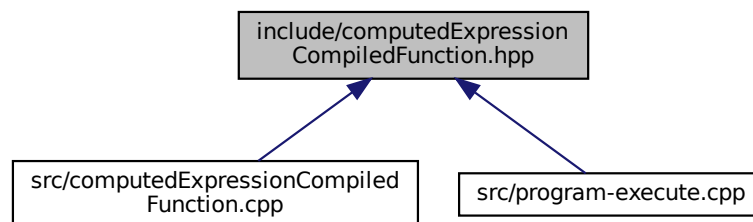
Declare the [Tang::ComputedExpressionCompiledFunction](#) class.

```
#include "computedExpression.hpp"
```

Include dependency graph for computedExpressionCompiledFunction.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionCompiledFunction](#)
Represents a Compiled Function declared in the script.

6.30.1 Detailed Description

Declare the [Tang::ComputedExpressionCompiledFunction](#) class.

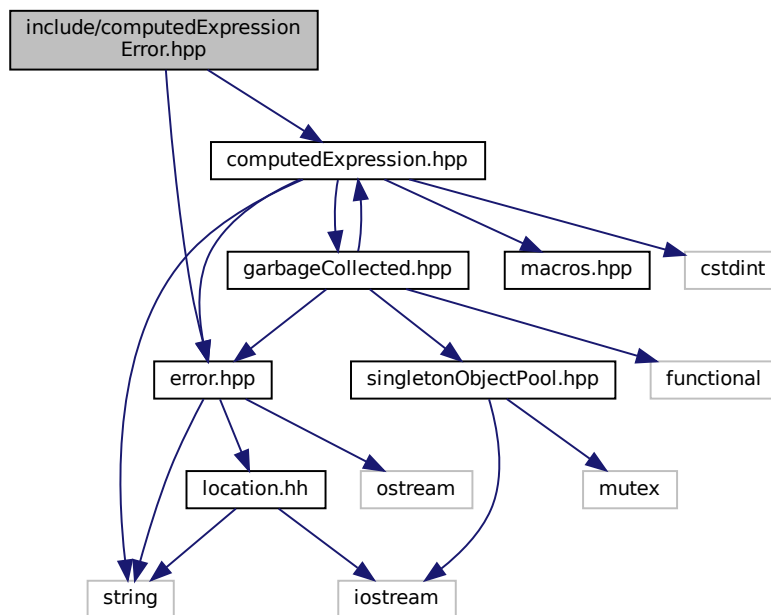
6.31 include/computedExpressionError.hpp File Reference

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](#).

6.31.1 Detailed Description

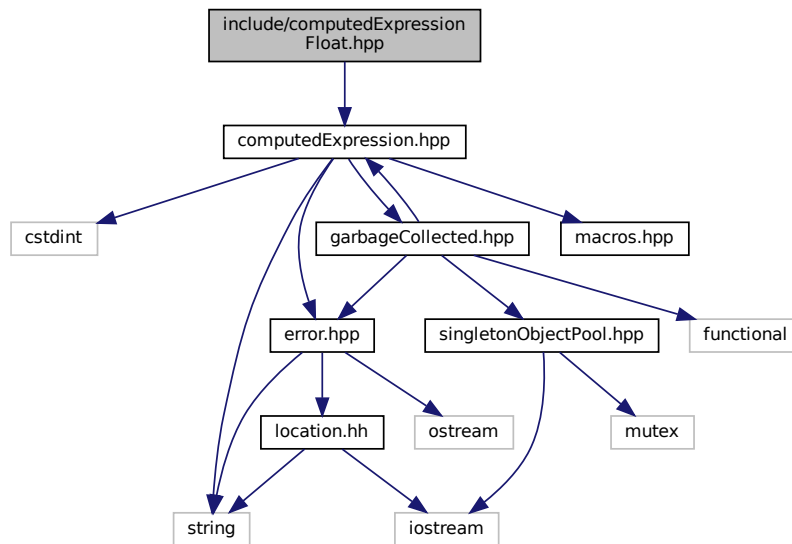
Declare the [Tang::ComputedExpressionError](#) class.

6.32 include/computedExpressionFloat.hpp File Reference

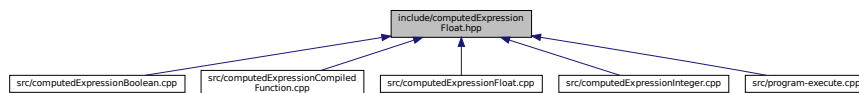
Declare the [Tang::ComputedExpressionFloat](#) class.

```
#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.

6.32.1 Detailed Description

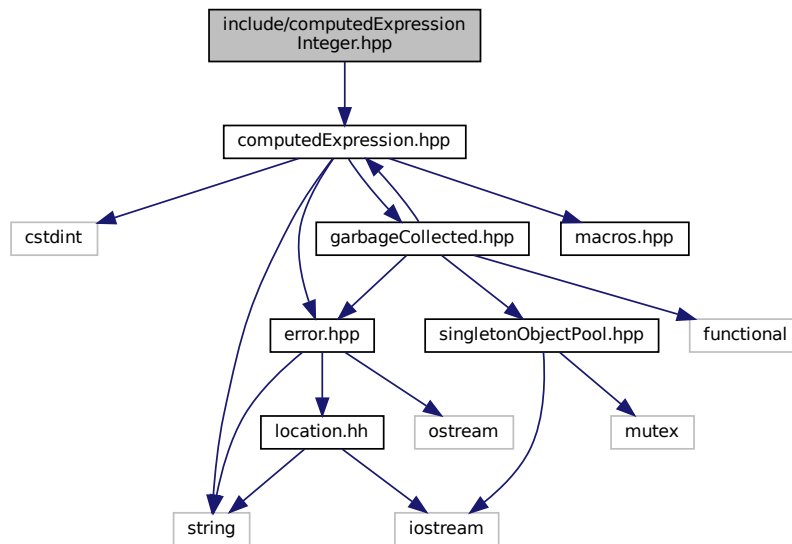
Declare the [Tang::ComputedExpressionFloat](#) class.

6.33 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.

6.33.1 Detailed Description

Declare the [Tang::ComputedExpressionInteger](#) class.

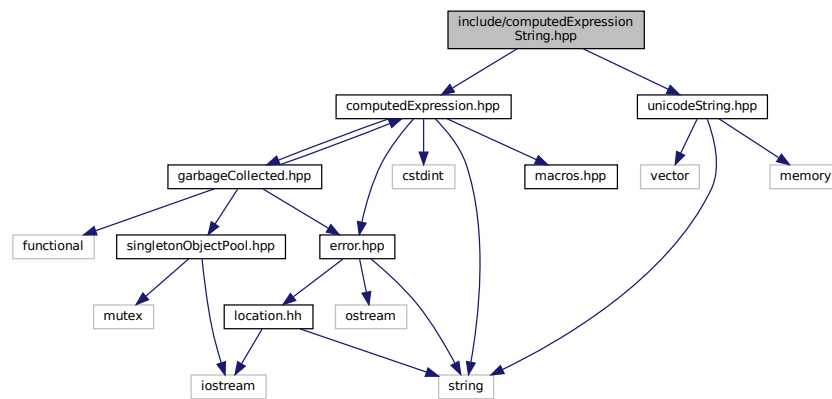
6.34 include/computedExpressionString.hpp File Reference

Declare the [Tang::ComputedExpressionString](#) class.

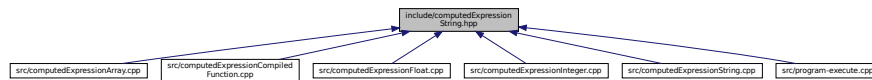
```
#include "computedExpression.hpp"
```

```
#include "unicodeString.hpp"
```

Include dependency graph for computedExpressionString.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionString](#)
Represents a String that is the result of a computation.

6.34.1 Detailed Description

Declare the [Tang::ComputedExpressionString](#) class.

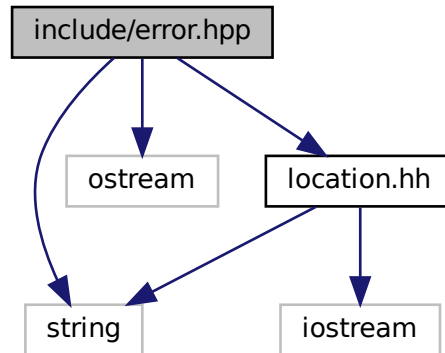
6.35 include/error.hpp File Reference

Declare the [Tang::Error](#) class used to describe syntax and runtime errors.

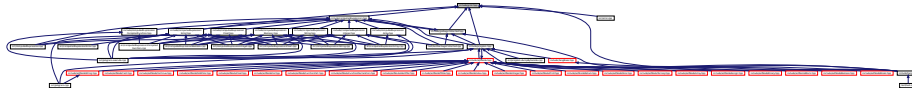
```
#include <string>
#include <ostream>
```

```
#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.

6.35.1 Detailed Description

Declare the [Tang::Error](#) class used to describe syntax and runtime errors.

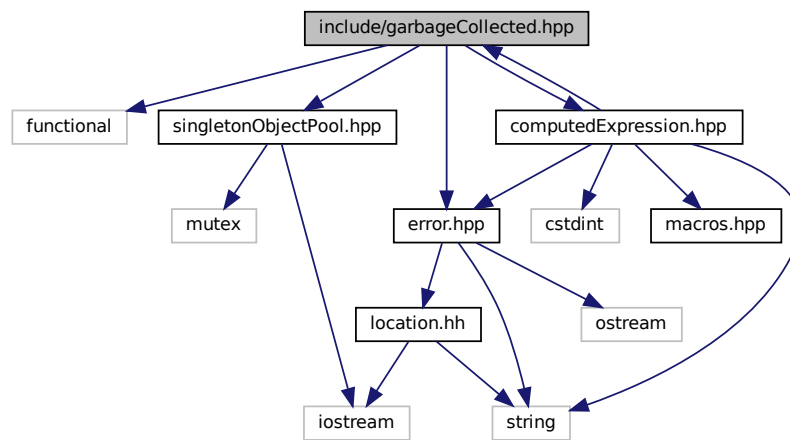
6.36 include/garbageCollected.hpp File Reference

Declare the [Tang::GarbageCollected](#) class.

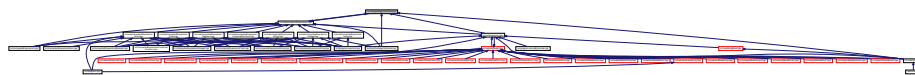
```
#include <functional>
#include "singletonObjectPool.hpp"
#include "computedExpression.hpp"
```

```
#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.

6.36.1 Detailed Description

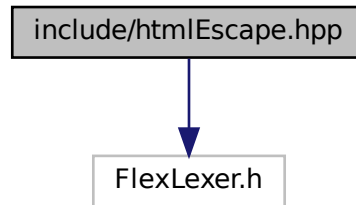
Declare the [Tang::GarbageCollected](#) class.

6.37 include/htmlEscape.hpp File Reference

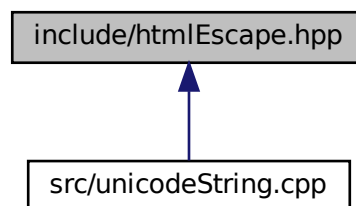
Declare the [Tang::HtmlEscape](#) used to tokenize a Tang script.

```
#include <FlexLexer.h>
```

Include dependency graph for htmlEscape.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::HtmlEscape](#)

The Flex lexer class for the main Tang language.

Macros

- #define **yyFlexLexer** TangHtmlEscapeFlexLexer
- #define **YY_DECL** std::string [Tang::HtmlEscape::get_next_token\(\)](#)

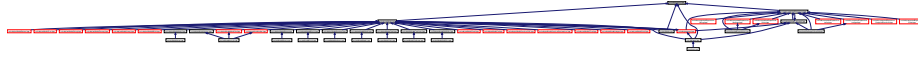
6.37.1 Detailed Description

Declare the [Tang::HtmlEscape](#) used to tokenize a Tang script.

6.38 include/macros.hpp File Reference

Contains generic macros.

This graph shows which files directly or indirectly include this file:



Typedefs

- using `Tang::integer_t` = `int32_t`
Define the size of signed integers used by Tang.
- using `Tang::uinteger_t` = `int32_t`
Define the size of integers used by Tang.
- using `Tang::float_t` = `float`
Define the size of floats used by Tang.

6.38.1 Detailed Description

Contains generic macros.

6.39 include/opcode.hpp File Reference

Declare the Opcodes used in the Bytecode representation of a program.

This graph shows which files directly or indirectly include this file:



Enumerations

- enum class `Tang::Opcode` {
`POP` , `PEEK` , `POKE` , `COPY` ,
`JMP` , `JMPF` , `JMPF_POP` , `JMPT` ,
`JMPT_POP` , `NULLVAL` , `INTEGER` , `FLOAT` ,
`BOOLEAN` , `STRING` , `ARRAY` , `FUNCTION` ,
`ASSIGNINDEX` , `ADD` , `SUBTRACT` , `MULTIPLY` ,
`DIVIDE` , `MODULO` , `NEGATIVE` , `NOT` ,
`LT` , `LTE` , `GT` , `GTE` ,
`EQ` , `NEQ` , `INDEX` , `SLICE` ,
`CASTINTEGER` , `CASTFLOAT` , `CASTBOOLEAN` , `CALLFUNC` ,
`RETURN` , `PRINT` }

6.39.1 Detailed Description

Declare the Opcodes used in the Bytecode representation of a program.

6.39.2 Enumeration Type Documentation

6.39.2.1 Opcode

```
enum Tang::Opcode [strong]
```

Enumerator

POP	Pop a val.
PEEK	Stack # (from fp): push val from stack #.
POKE	Stack # (from fp): Copy a val, store @ stack #.
COPY	Stack # (from fp): Deep copy val @ stack #, store @ stack #.
JMP	PC #: set pc to PC #.
JMPF	PC #: read val, if false, set pc to PC #.
JMPF_POP	PC #: pop val, if false, set pc to PC #.
JMPT	PC #: read val, if true, set pc to PC #.
JMPT_POP	PC #: pop val, if true, set pc to PC #.
NULLVAL	Push a null onto the stack.
INTEGER	Push an integer onto the stack.
FLOAT	Push a floating point number onto the stack.
BOOLEAN	Push a boolean onto the stack.
STRING	Get len, char string: push string.
ARRAY	Get len, pop len items, putting them into an array with the last array item popped first.
FUNCTION	Get argc, PC#: push function(argc, PC #)
ASSIGNINDEX	Pop index, pop collection, pop value, push (collection[index] = value)
ADD	Pop rhs, pop lhs, push lhs + rhs.
SUBTRACT	Pop rhs, pop lhs, push lhs - rhs.
MULTIPLY	Pop rhs, pop lhs, push lhs * rhs.
DIVIDE	Pop rhs, pop lhs, push lhs / rhs.
MODULO	Pop rhs, pop lhs, push lhs % rhs.
NEGATIVE	Pop val, push negative val.
NOT	Pop val, push logical not of val.
LT	Pop rhs, pop lhs, push lhs < rhs.
LTE	Pop rhs, pop lhs, push lhs <= rhs.
GT	Pop rhs, pop lhs, push lhs > rhs.
GTE	Pop rhs, pop lhs, push lhs >= rhs.
EQ	Pop rhs, pop lhs, push lhs == rhs.
NEQ	Pop rhs, pop lhs, push lhs != rhs.
INDEX	Pop index, pop collection, push collection[index].
SLICE	Pop skip, pop end, pop begin, pop collection, push collection[begin:end:skip].
CASTINTEGER	Pop a val, typecast to int, push.

Enumerator

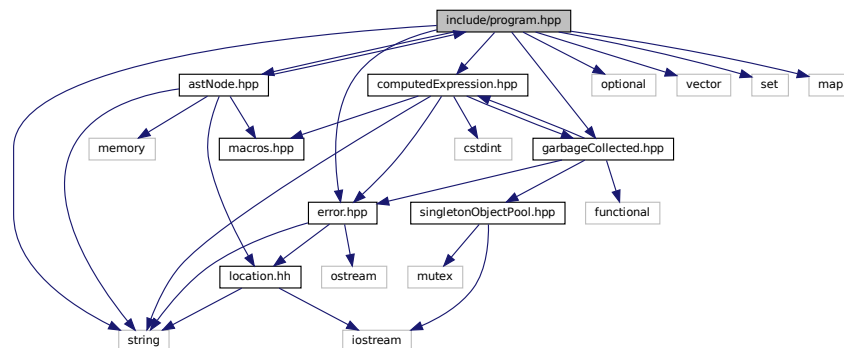
CASTFLOAT	Pop a val, typecast to float, push.
CASTBOOLEAN	Pop a val, typecast to boolean, push.
CALLFUNC	Get argc, Pop a function, execute function if argc matches.
RETURN	Get stack #, pop return val, pop (stack #) times, push val, restore fp, restore pc.
PRINT	Pop val, print(val), push error or NULL.

6.40 include/program.hpp File Reference

Declare the [Tang::Program](#) class used to compile and execute source code.

```
#include <string>
#include <optional>
#include <vector>
#include <set>
#include <map>
#include "astNode.hpp"
#include "error.hpp"
#include "computedExpression.hpp"
#include "garbageCollected.hpp"
```

Include dependency graph for program.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::Program](#)
Represents a compiled script or template that may be executed.

Typedefs

- using `Tang::Bytecode` = `std::vector< Tang::uinteger_t >`
Contains the Opcodes of a compiled program.

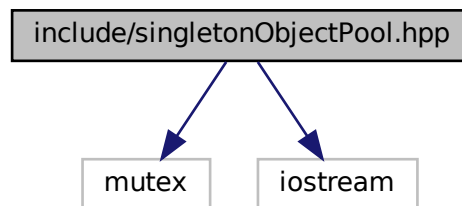
6.40.1 Detailed Description

Declare the `Tang::Program` class used to compile and execute source code.

6.41 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.

6.41.1 Detailed Description

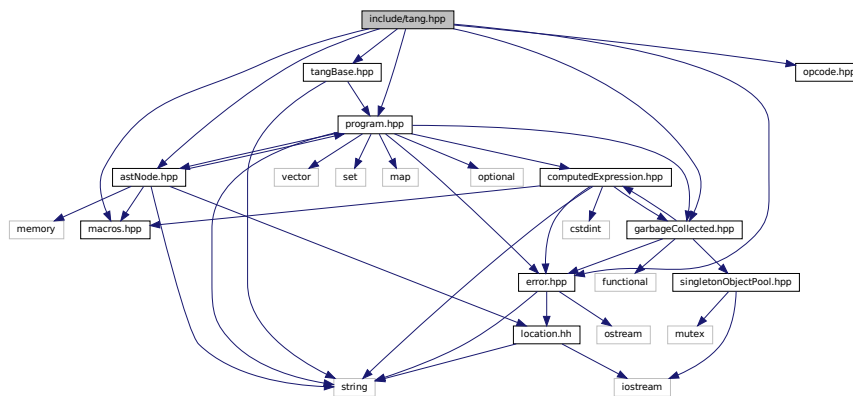
Declare the `Tang::SingletonObjectPool` class.

6.42 include/tang.hpp File Reference

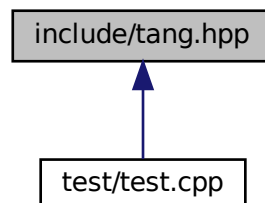
Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

```
#include "macros.hpp"
#include "tangBase.hpp"
#include "astNode.hpp"
#include "error.hpp"
#include "garbageCollected.hpp"
#include "program.hpp"
#include "opcode.hpp"
```

Include dependency graph for tang.hpp:



This graph shows which files directly or indirectly include this file:



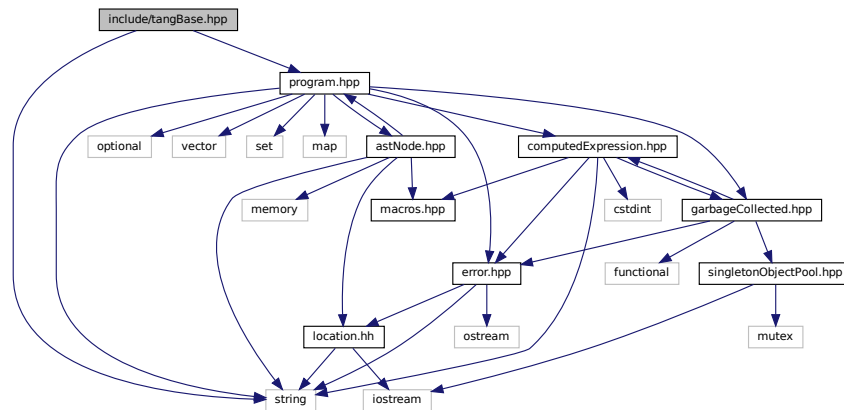
6.42.1 Detailed Description

Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

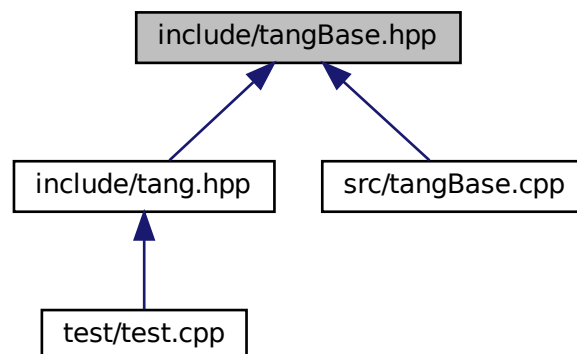
6.43 include/tangBase.hpp File Reference

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.

6.43.1 Detailed Description

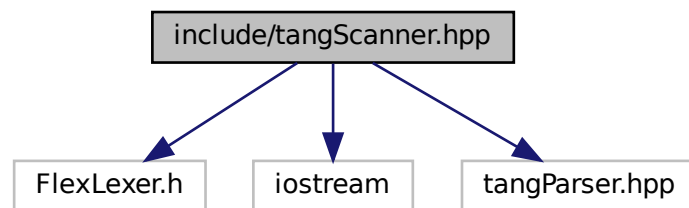
Declare the [Tang::TangBase](#) class used to interact with Tang.

6.44 include/tangScanner.hpp File Reference

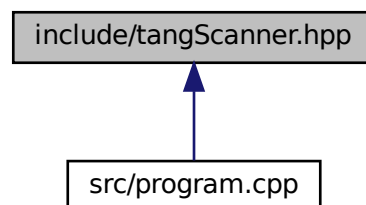
Declare the [Tang::TangScanner](#) used to tokenize a Tang script.

```
#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\(\)](#)

6.44.1 Detailed Description

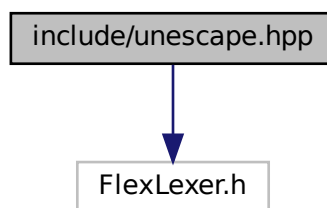
Declare the [Tang::TangScanner](#) used to tokenize a Tang script.

6.45 include/unescape.hpp File Reference

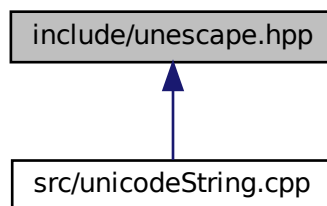
Declare the [Tang::Unescape](#) used to tokenize a Tang script.

```
#include <FlexLexer.h>
```

Include dependency graph for unescape.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::Unescape](#)
The Flex lexer class for the main Tang language.

Macros

- #define **yyFlexLexer** TangUnescapeFlexLexer
- #define **YY_DECL** std::string [Tang::Unescape::get_next_token\(\)](#)

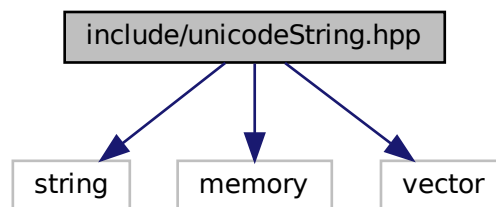
6.45.1 Detailed Description

Declare the [Tang::Unescape](#) used to tokenize a Tang script.

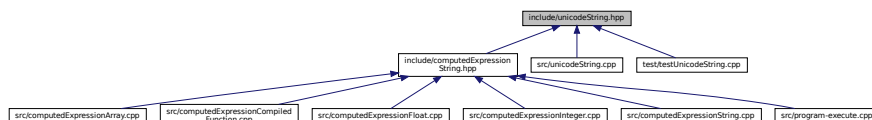
6.46 include/unicodeString.hpp File Reference

Contains the code to interface with the ICU library.

```
#include <string>
#include <memory>
#include <vector>
Include dependency graph for unicodeString.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::UnicodeString](#)
Represents a UTF-8 encoded string that is Unicode-aware.

Functions

- `std::string Tang::unescape (const std::string &str)`
Return an "unescape" version of the provided string, which, when interpreted by Tang, should result in a representation equivalent to the original source string.
- `std::string Tang::htmlEscape (const std::string &str)`
Return an "html escaped" version of the provided string.

6.46.1 Detailed Description

Contains the code to interface with the ICU library.

6.46.2 Function Documentation

6.46.2.1 `htmlEscape()`

```
string Tang::htmlEscape (  
    const std::string & str )
```

Return an "html escaped" version of the provided string.

Only "critical" characters `<`, `>`, `&`, `"`, and ``` will be escaped. All other characters will be allowed through unaltered. The result is a UTF-8 encoded string that is safe for inclusion in an HTML template without disturbing the HTML structure.

Parameters

<i>str</i>	The string to be escaped.
------------	---------------------------

Returns

An "escaped" version of the provided string.

Here is the call graph for this function:



6.46.2.2 `unescape()`

```
string Tang::unescape (  
    const std::string & str )
```

Return an "unescaped" version of the provided string, which, when interpreted by Tang, should result in a representation equivalent to the original source string.

Parameters

<i>str</i>	The string to be unescaped.
------------	-----------------------------

Returns

An "unescaped" version of the provided string.

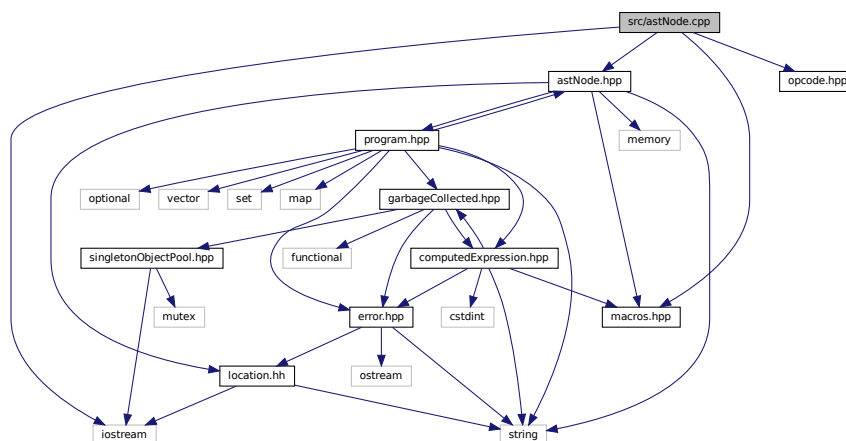
Here is the call graph for this function:



6.47 src/astNode.cpp File Reference

Define the [Tang::AstNode](#) class.

```
#include <iostream>
#include "macros.hpp"
#include "astNode.hpp"
#include "opcode.hpp"
Include dependency graph for astNode.cpp:
```



6.47.1 Detailed Description

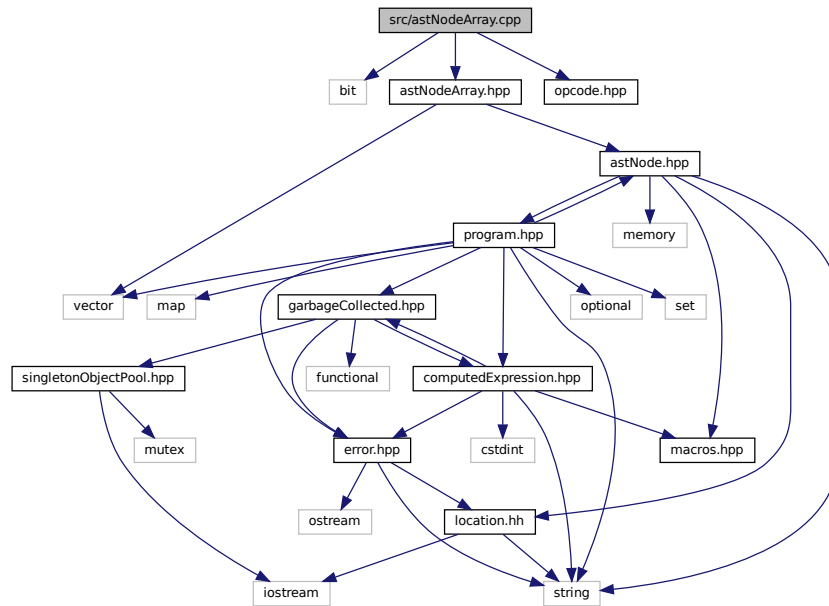
Define the [Tang::AstNode](#) class.

6.48 src/astNodeArray.cpp File Reference

Define the [Tang::AstNodeArray](#) class.

```
#include <bit>
#include "astNodeArray.hpp"
#include "opcode.hpp"
```

Include dependency graph for astNodeArray.cpp:



6.48.1 Detailed Description

Define the [Tang::AstNodeArray](#) class.

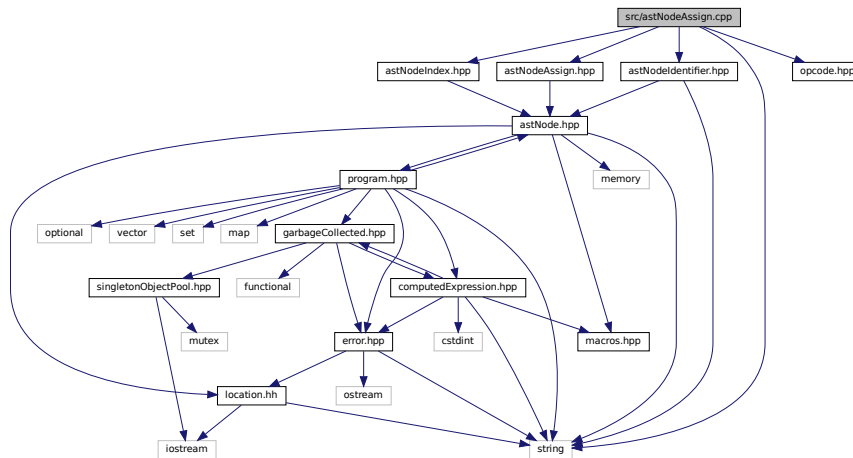
6.49 src/astNodeAssign.cpp File Reference

Define the [Tang::AstNodeAssign](#) class.

```
#include <string>
#include "astNodeAssign.hpp"
#include "astNodeIdentifier.hpp"
#include "astNodeIndex.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeAssign.cpp:



6.49.1 Detailed Description

Define the [Tang::AstNodeAssign](#) class.

6.50 src/astNodeBinary.cpp File Reference

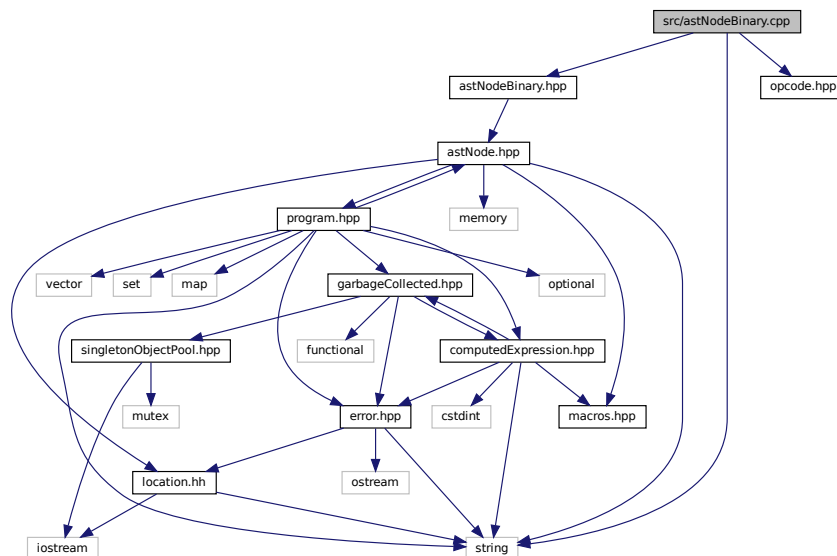
Define the [Tang::AstNodeBinary](#) class.

```
#include <string>
```

```
#include "astNodeBinary.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeBinary.cpp:



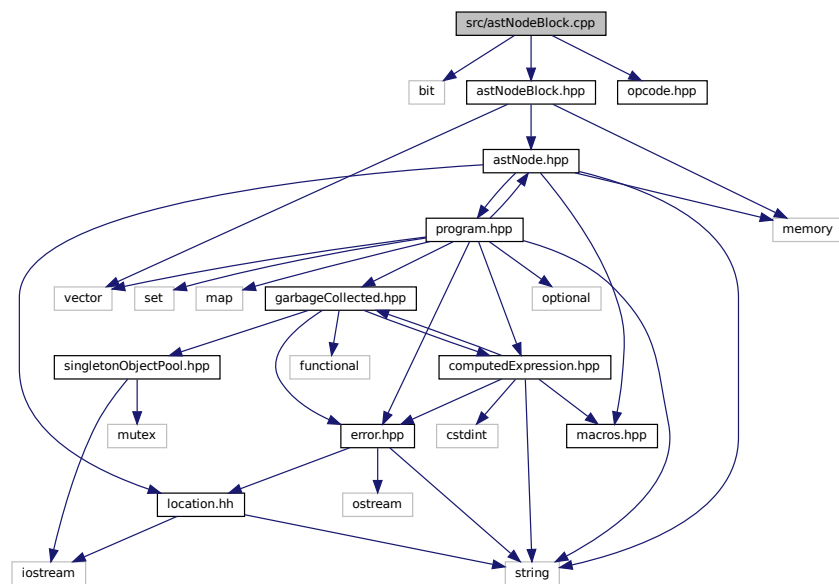
6.50.1 Detailed Description

Define the [Tang::AstNodeBinary](#) class.

6.51 src/astNodeBlock.cpp File Reference

Define the [Tang::AstNodeBlock](#) class.

```
#include <bit>
#include "astNodeBlock.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeBlock.cpp:
```



6.51.1 Detailed Description

Define the [Tang::AstNodeBlock](#) class.

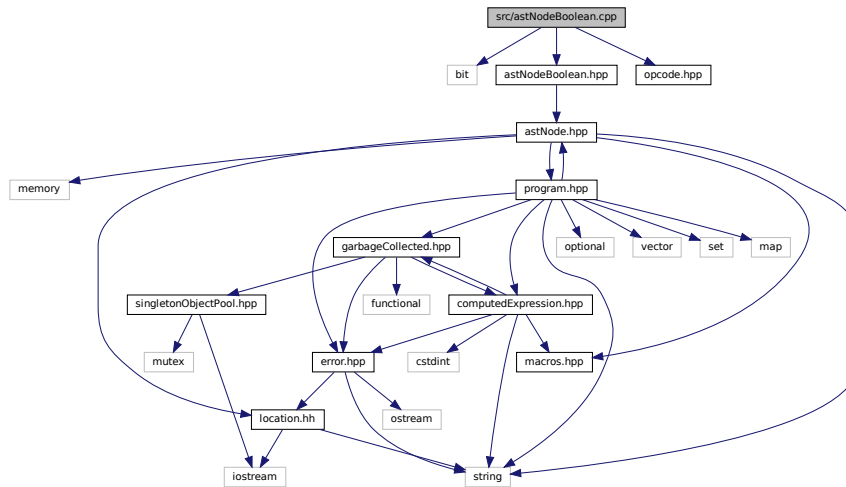
6.52 src/astNodeBoolean.cpp File Reference

Define the [Tang::AstNodeBoolean](#) class.

```
#include <bit>
#include "astNodeBoolean.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeBoolean.cpp:



6.52.1 Detailed Description

Define the [Tang::AstNodeBoolean](#) class.

6.53 src/astNodeBreak.cpp File Reference

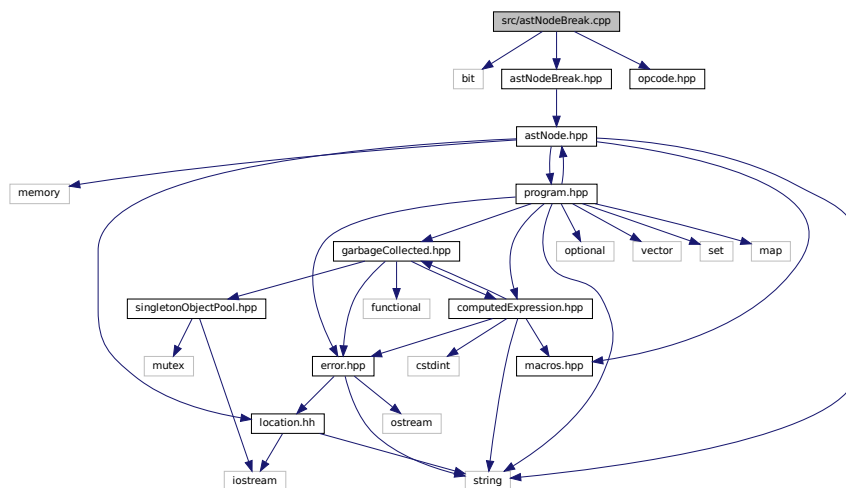
Define the [Tang::AstNodeBreak](#) class.

```
#include <bit>
```

```
#include "astNodeBreak.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeBreak.cpp:



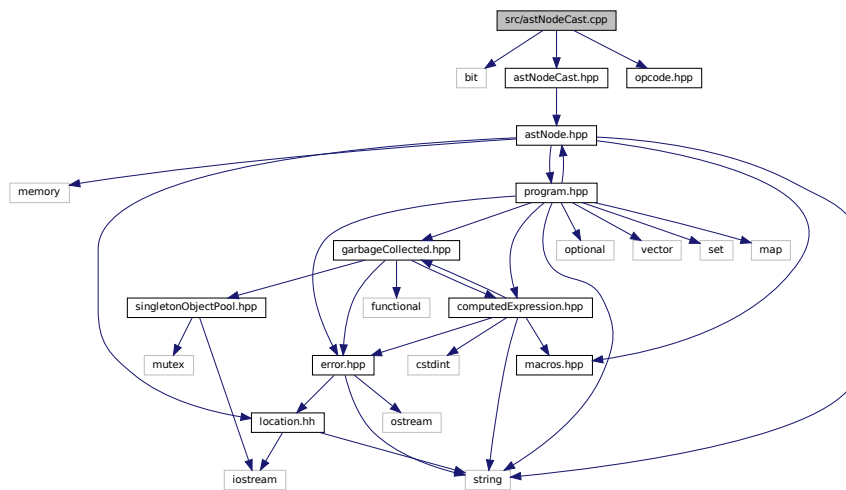
6.53.1 Detailed Description

Define the [Tang::AstNodeBreak](#) class.

6.54 src/astNodeCast.cpp File Reference

Define the [Tang::AstNodeCast](#) class.

```
#include <bit>
#include "astNodeCast.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeCast.cpp:
```



6.54.1 Detailed Description

Define the [Tang::AstNodeCast](#) class.

6.55 src/astNodeContinue.cpp File Reference

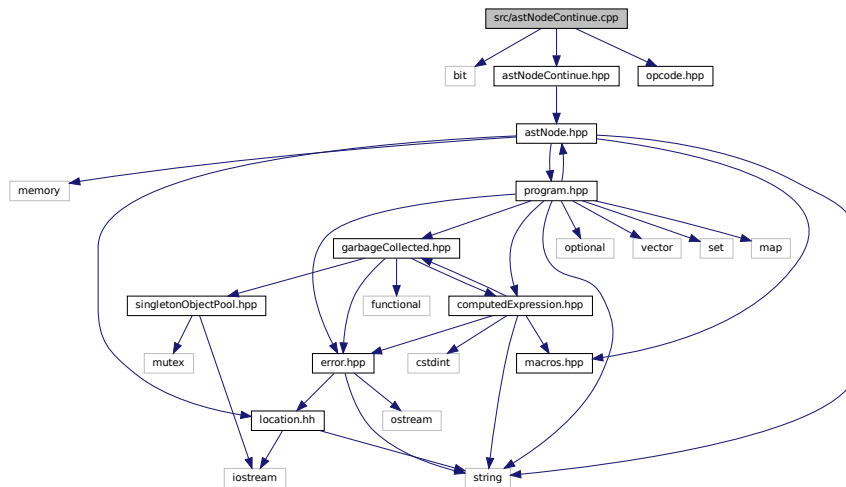
Define the [Tang::AstNodeContinue](#) class.

```
#include <bit>
#include "astNodeContinue.hpp"
```



```
#include "opcode.hpp"
```

Include dependency graph for astNodeContinue.cpp:



6.55.1 Detailed Description

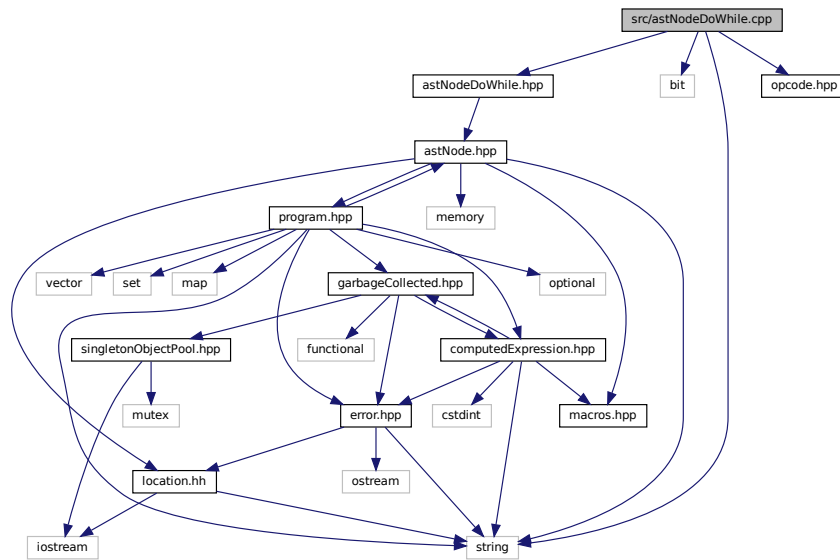
Define the [Tang::AstNodeContinue](#) class.

6.56 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`:



6.56.1 Detailed Description

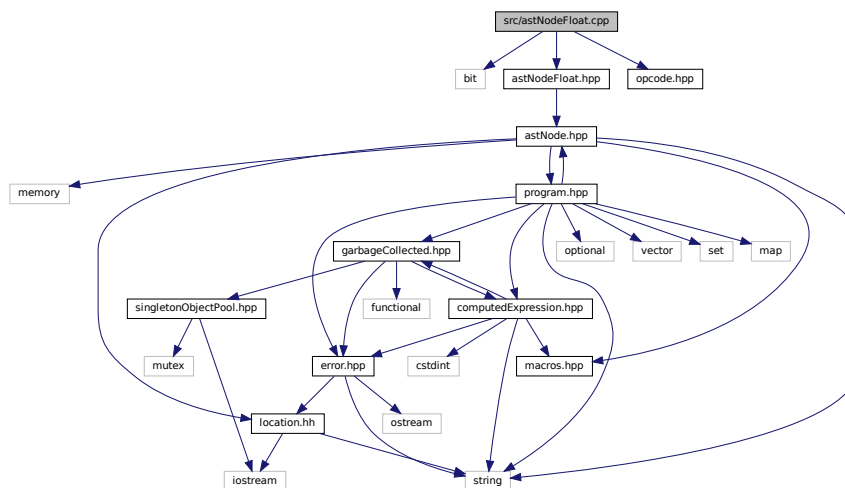
Define the [Tang::AstNodeDoWhile](#) class.

6.57 src/astNodeFloat.cpp File Reference

Define the [Tang::AstNodeFloat](#) class.

```
#include <bit>
#include "astNodeFloat.hpp"
#include "opcode.hpp"
```

Include dependency graph for `astNodeFloat.cpp`:



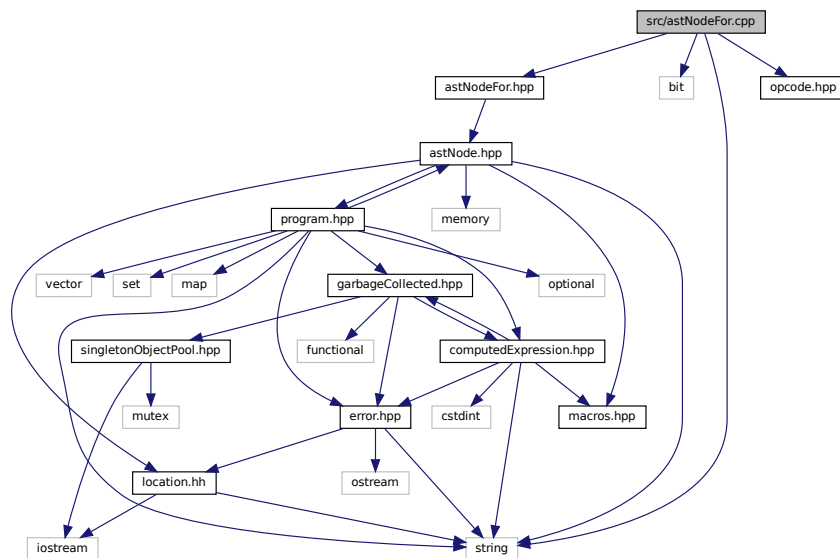
6.57.1 Detailed Description

Define the [Tang::AstNodeFloat](#) class.

6.58 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:
```



6.58.1 Detailed Description

Define the [Tang::AstNodeFor](#) class.

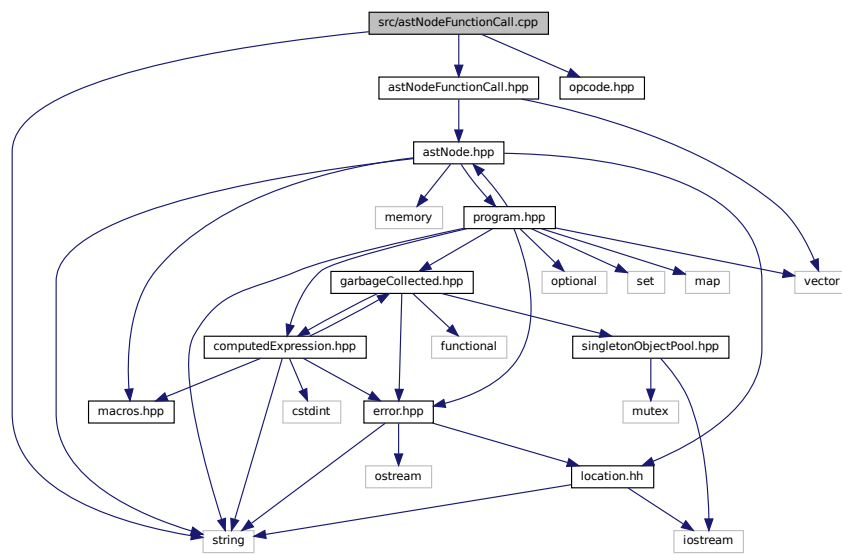
6.59 src/astNodeFunctionCall.cpp File Reference

Define the [Tang::AstNodeFunctionCall](#) class.

```
#include <string>
#include "astNodeFunctionCall.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for `astNodeFunctionCall.cpp`:



6.59.1 Detailed Description

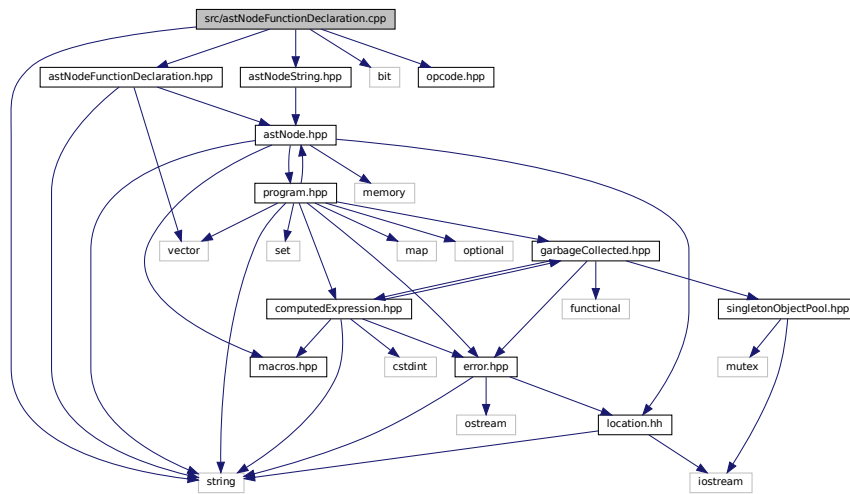
Define the [Tang::AstNodeFunctionCall](#) class.

6.60 src/astNodeFunctionDeclaration.cpp File Reference

Define the [Tang::AstNodeFunctionDeclaration](#) class.

```
#include <string>
#include <bit>
#include "astNodeFunctionDeclaration.hpp"
#include "astNodeString.hpp"
#include "opcode.hpp"
```

Include dependency graph for astNodeFunctionDeclaration.cpp:



6.60.1 Detailed Description

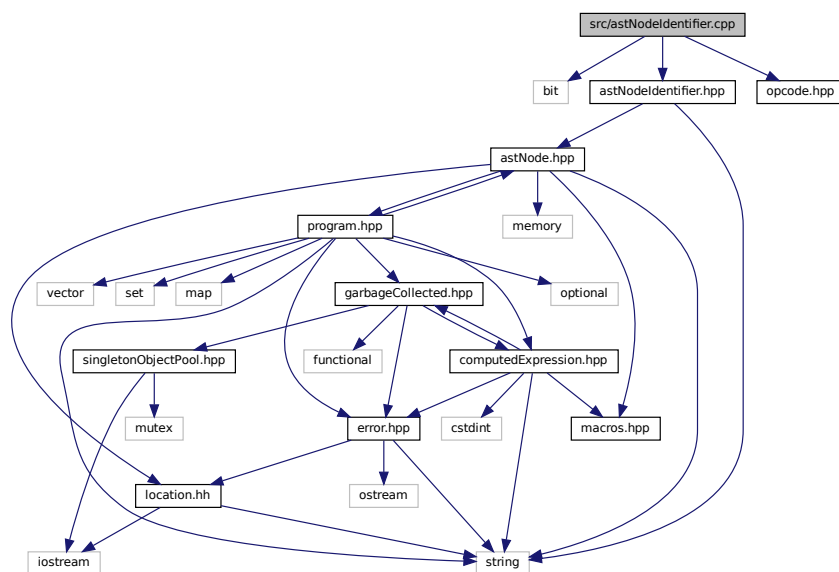
Define the [Tang::AstNodeFunctionDeclaration](#) class.

6.61 src/astNodeIdentifier.cpp File Reference

Define the [Tang::AstNodeIdentifier](#) class.

```
#include <bit>
#include "astNodeIdentifier.hpp"
#include "opcode.hpp"
```

Include dependency graph for astNodeIdentifier.cpp:



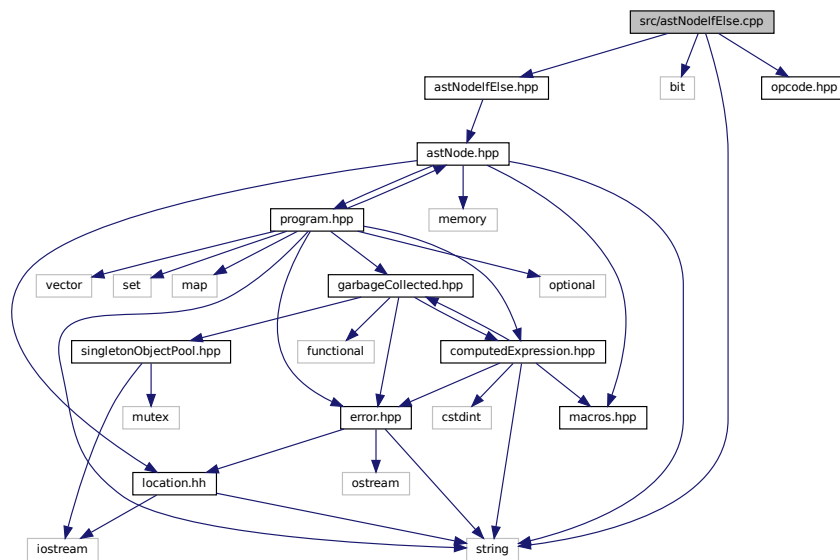
6.61.1 Detailed Description

Define the [Tang::AstNodeIdentifier](#) class.

6.62 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:
```



6.62.1 Detailed Description

Define the [Tang::AstNodeIfElse](#) class.

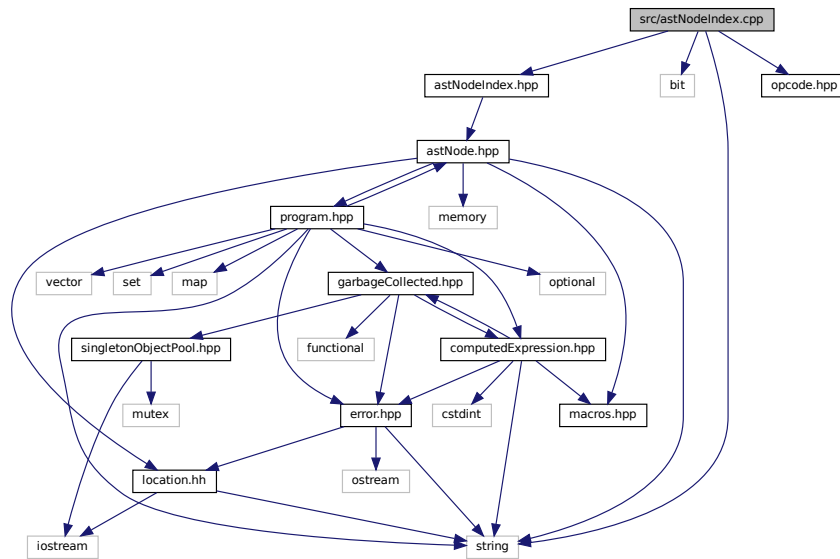
6.63 src/astNodeIndex.cpp File Reference

Define the [Tang::AstNodeIndex](#) class.

```
#include <string>
#include <bit>
#include "astNodeIndex.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeInteger.cpp:



6.63.1 Detailed Description

Define the [Tang::AstNodeIndex](#) class.

6.64 src/astNodeInteger.cpp File Reference

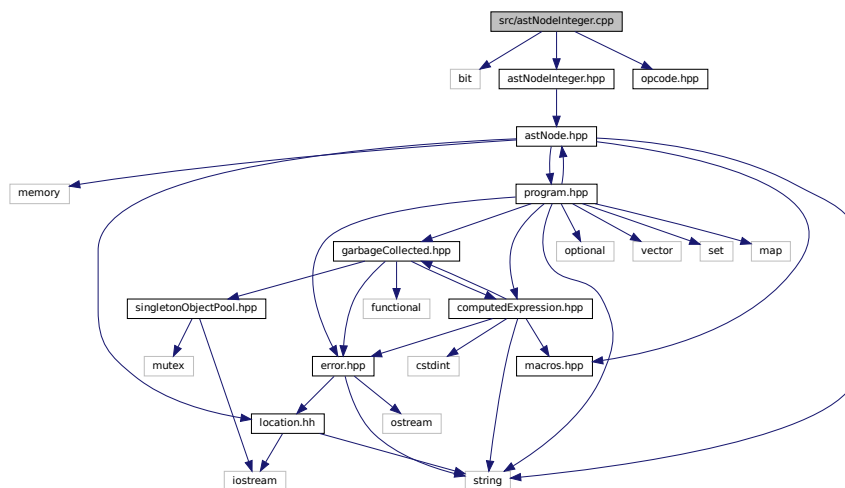
Define the [Tang::AstNodeInteger](#) class.

```
#include <bit>
```

```
#include "astNodeInteger.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeInteger.cpp:



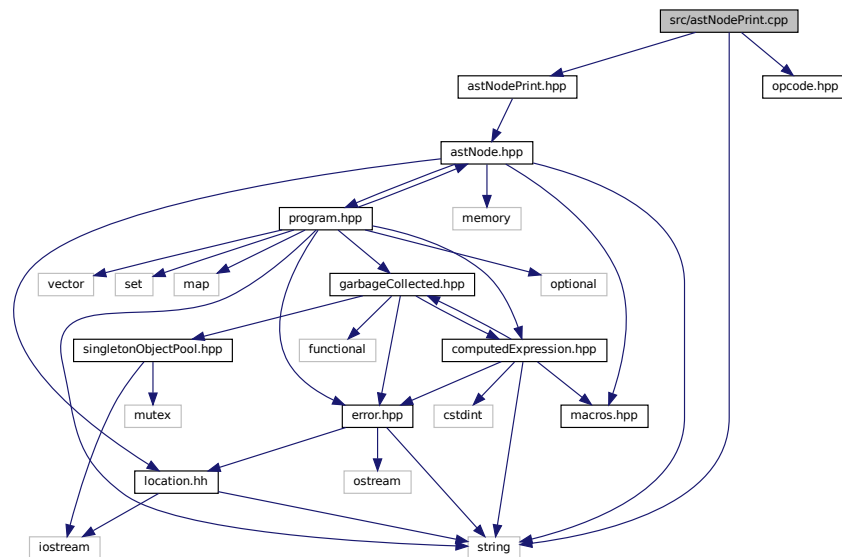
6.64.1 Detailed Description

Define the [Tang::AstNodeInteger](#) class.

6.65 src/astNodePrint.cpp File Reference

Define the [Tang::AstNodePrint](#) class.

```
#include <string>
#include "astNodePrint.hpp"
#include "opcode.hpp"
Include dependency graph for astNodePrint.cpp:
```



6.65.1 Detailed Description

Define the [Tang::AstNodePrint](#) class.

6.66 src/astNodeReturn.cpp File Reference

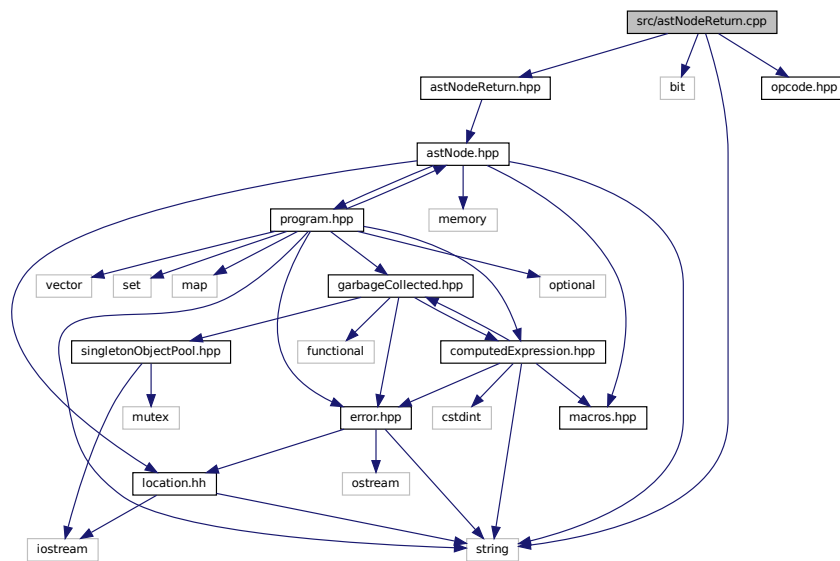
Define the [Tang::AstNodeReturn](#) class.

```
#include <string>
#include <bit>
#include "astNodeReturn.hpp"
```



```
#include "opcode.hpp"
```

Include dependency graph for astNodeReturn.cpp:



6.66.1 Detailed Description

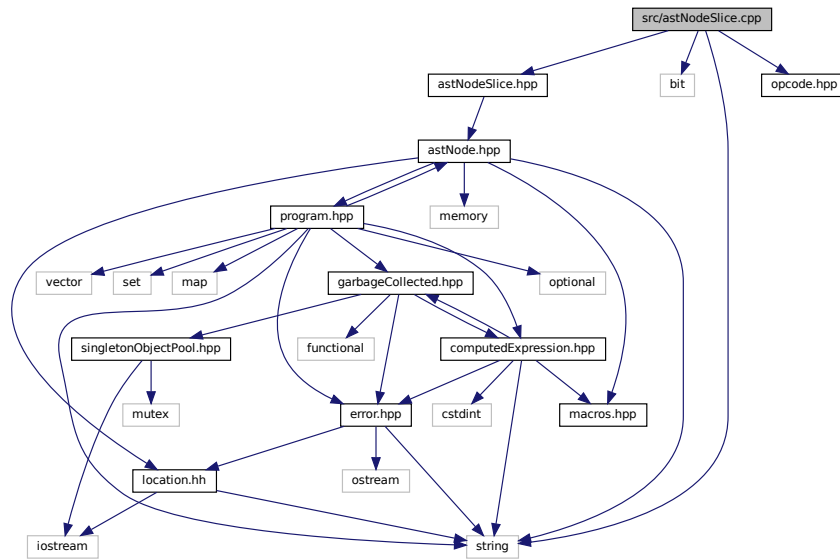
Define the [Tang::AstNodeReturn](#) class.

6.67 src/astNodeSlice.cpp File Reference

Define the [Tang::AstNodeSlice](#) class.

```
#include <string>
#include <bit>
#include "astNodeSlice.hpp"
#include "opcode.hpp"
```

Include dependency graph for astNodeSlice.cpp:



6.67.1 Detailed Description

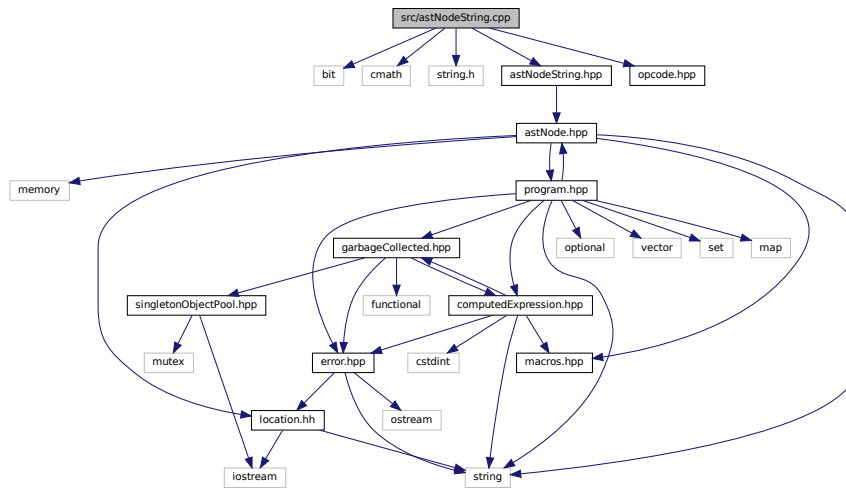
Define the [Tang::AstNodeSlice](#) class.

6.68 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:



6.68.1 Detailed Description

Define the [Tang::AstNodeString](#) class.

6.69 src/astNodeTernary.cpp File Reference

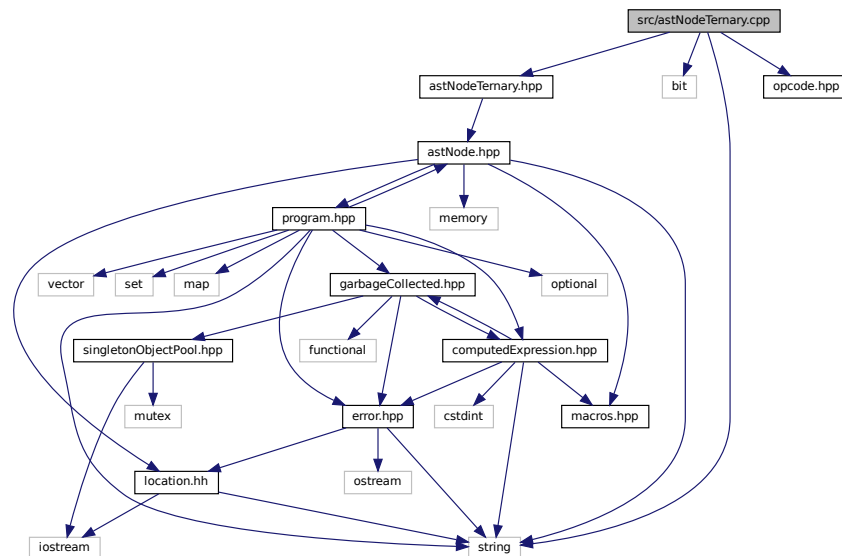
Define the [Tang::AstNodeTernary](#) class.

```

#include <string>
#include <bit>
#include "astNodeTernary.hpp"
#include "opcode.hpp"

```

Include dependency graph for astNodeTernary.cpp:



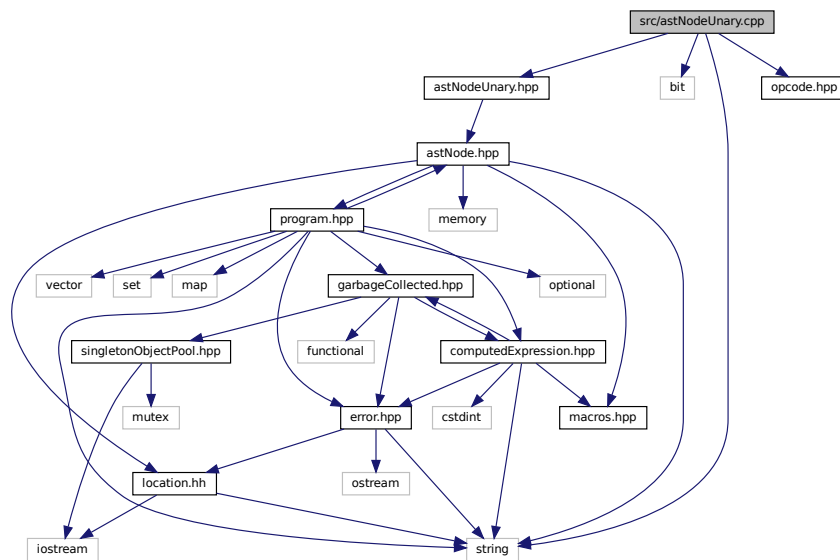
6.69.1 Detailed Description

Define the [Tang::AstNodeUnary](#) class.

6.70 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:
```



6.70.1 Detailed Description

Define the [Tang::AstNodeUnary](#) class.

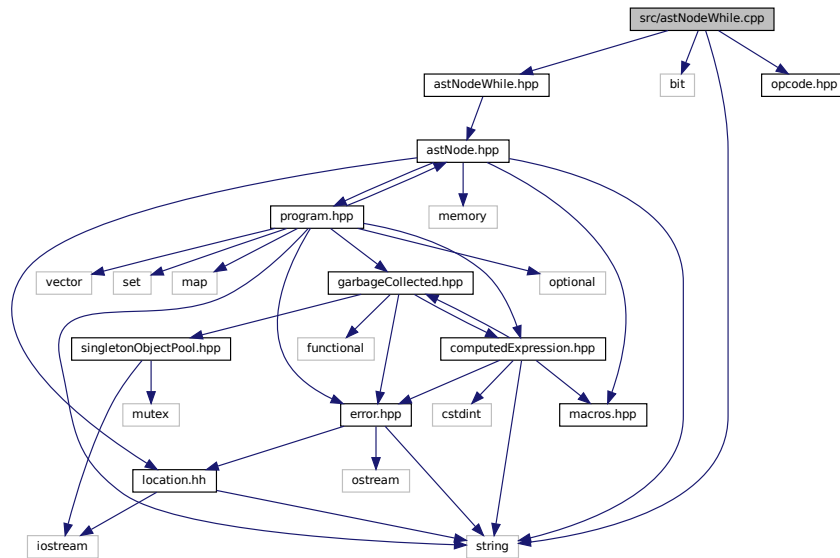
6.71 src/astNodeWhile.cpp File Reference

Define the [Tang::AstNodeWhile](#) class.

```
#include <string>
#include <bit>
#include "astNodeWhile.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeWhile.cpp:



6.71.1 Detailed Description

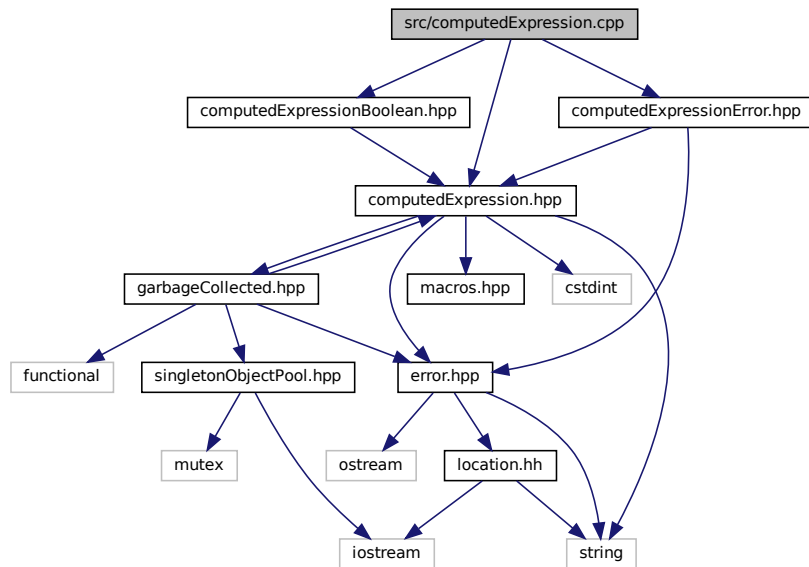
Define the [Tang::AstNodeWhile](#) class.

6.72 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`:



6.72.1 Detailed Description

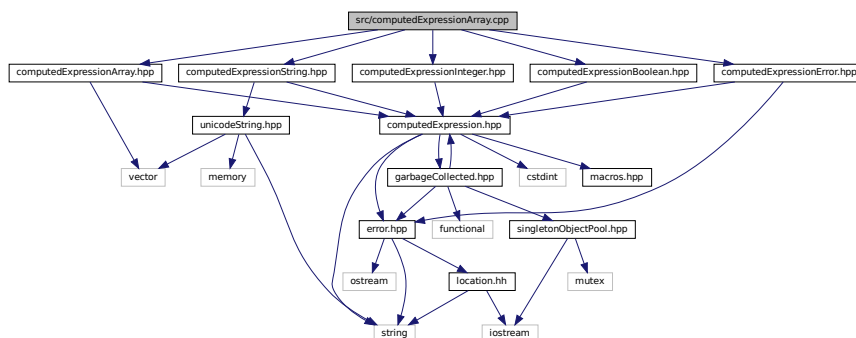
Define the [Tang::ComputedExpression](#) class.

6.73 src/computedExpressionArray.cpp File Reference

Define the [Tang::ComputedExpressionArray](#) class.

```
#include "computedExpressionArray.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionError.hpp"
```

Include dependency graph for `computedExpressionArray.cpp`:



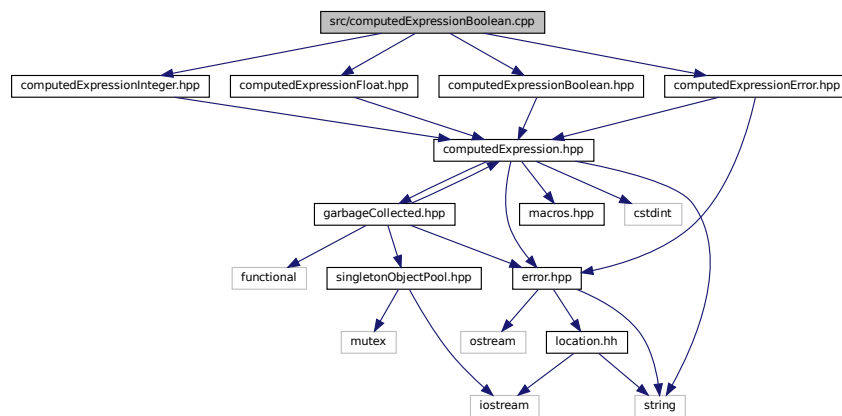
6.73.1 Detailed Description

Define the [Tang::ComputedExpressionArray](#) class.

6.74 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:
```



6.74.1 Detailed Description

Define the [Tang::ComputedExpressionBoolean](#) class.

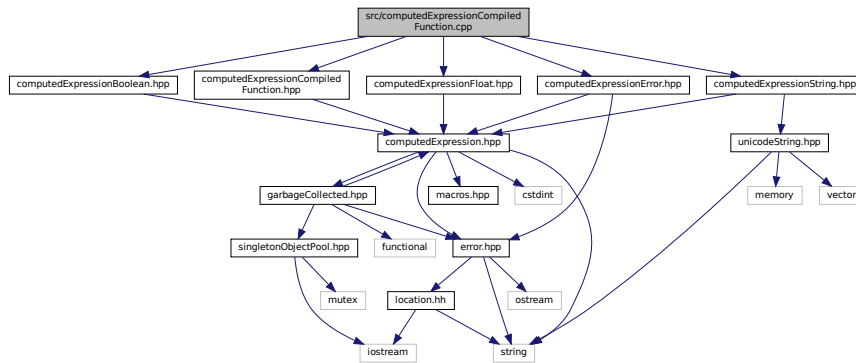
6.75 src/computedExpressionCompiledFunction.cpp File Reference

Define the [Tang::ComputedExpressionCompiledFunction](#) class.

```
#include "computedExpressionCompiledFunction.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
```

```
#include "computedExpressionError.hpp"
```

Include dependency graph for `computedExpressionCompiledFunction.cpp`:



6.75.1 Detailed Description

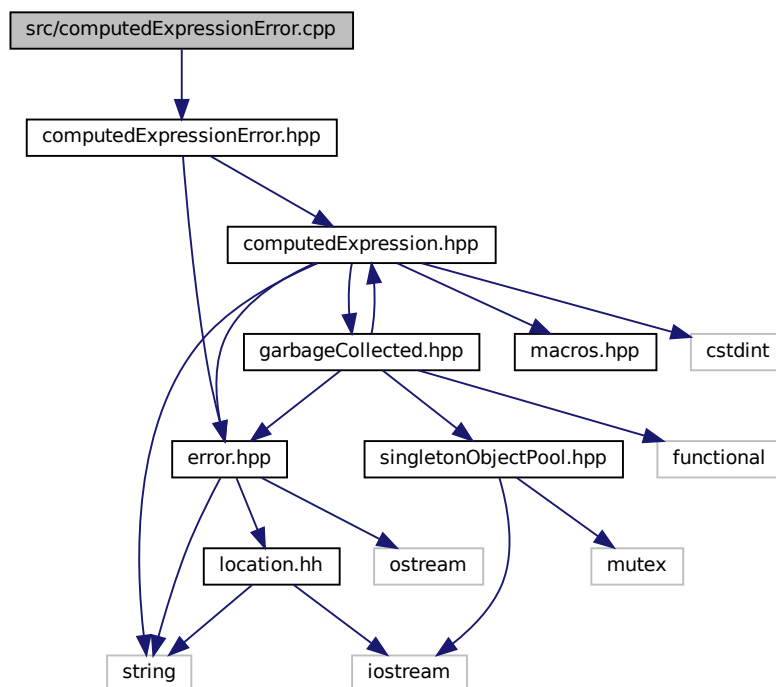
Define the [Tang::ComputedExpressionCompiledFunction](#) class.

6.76 src/computedExpressionError.cpp File Reference

Define the [Tang::ComputedExpressionError](#) class.

```
#include "computedExpressionError.hpp"
```

Include dependency graph for `computedExpressionError.cpp`:



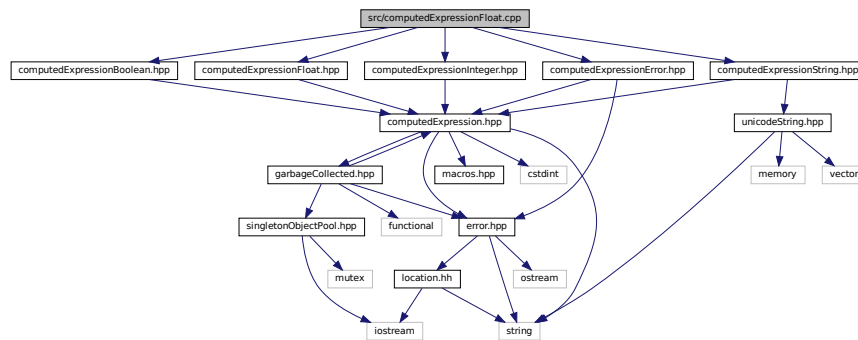
6.76.1 Detailed Description

Define the [Tang::ComputedExpressionError](#) class.

6.77 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:
```



6.77.1 Detailed Description

Define the [Tang::ComputedExpressionFloat](#) class.

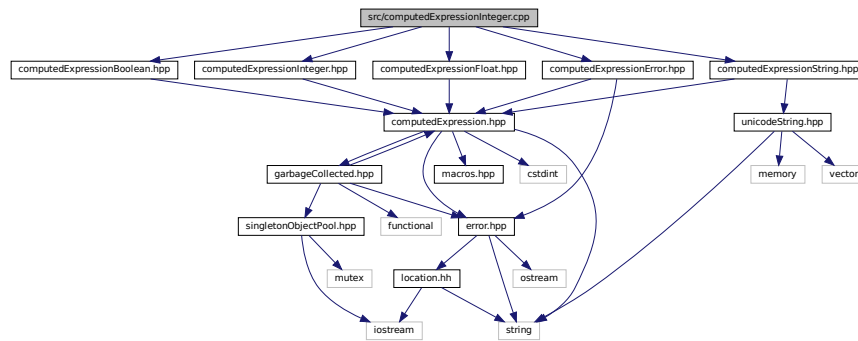
6.78 src/computedExpressionInteger.cpp File Reference

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:



6.78.1 Detailed Description

Define the [Tang::ComputedExpressionInteger](#) class.

6.79 src/computedExpressionString.cpp File Reference

Define the [Tang::ComputedExpressionString](#) class.

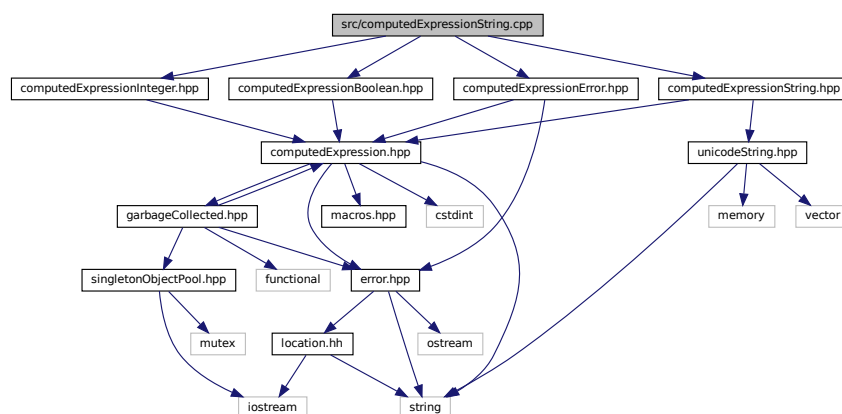
```
#include "computedExpressionString.hpp"
```

```
#include "computedExpressionBoolean.hpp"
```

```
#include "computedExpressionError.hpp"
```

```
#include "computedExpressionInteger.hpp"
```

Include dependency graph for computedExpressionString.cpp:



6.79.1 Detailed Description

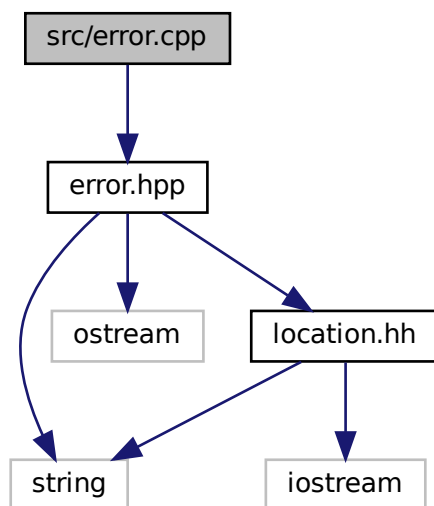
Define the [Tang::ComputedExpressionString](#) class.

6.80 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)`

6.80.1 Detailed Description

Define the [Tang::Error](#) class.

6.80.2 Function Documentation

6.80.2.1 `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

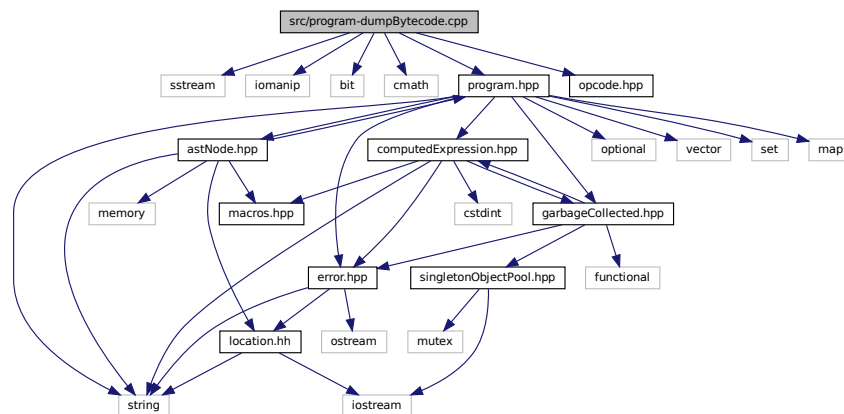
The output stream.

6.81 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.

6.81.1 Detailed Description

Define the [Tang::Program::dumpBytecode](#) method.

6.81.2 Macro Definition Documentation

6.81.2.1 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

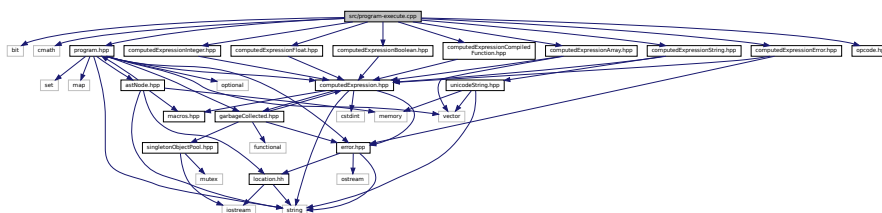
x	The number of additional vector entries that should exist.
---	--

6.82 src/program-execute.cpp File Reference

Define the `Tang::Program::execute` method.

```
#include <bit>
#include <cmath>
#include "program.hpp"
#include "opcode.hpp"
#include "computedExpressionError.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionArray.hpp"
#include "computedExpressionCompiledFunction.hpp"
```

Include dependency graph for program-execute.cpp:

**Macros**

- `#define EXECUTEPROGRAMCHECK(x)`
Verify the size of the Bytecode vector so that it may be safely accessed.
- `#define STACKCHECK(x)`
Verify the size of the stack vector so that it may be safely accessed.

6.82.1 Detailed Description

Define the [Tang::Program::execute](#) method.

6.82.2 Macro Definition Documentation

6.82.2.1 EXECUTEPROGRAMCHECK

```
#define EXECUTEPROGRAMCHECK(  
    x )
```

Value:

```
if (this->bytecode.size() < (pc + (x))) { \
    stack.push_back(GarbageCollected::make<ComputedExpressionError>(Error{"Opcode instruction  
truncated."})); \
    pc = this->bytecode.size(); \
    break; \
}
```

Verify the size of the Bytecode vector so that it may be safely accessed.

Parameters

x	The number of additional vector entries that should exist.
---	--

6.82.2.2 STACKCHECK

```
#define STACKCHECK(  
    x )
```

Value:

```
if (stack.size() < (fp + (x))) { \
    stack.push_back(GarbageCollected::make<ComputedExpressionError>(Error{"Insufficient stack depth."})); \
    pc = this->bytecode.size(); \
    break; \
}
```

Verify the size of the stack vector so that it may be safely accessed.

Parameters

x	The number of entries that should exist in the stack.
---	---

6.83 src/program.cpp File Reference

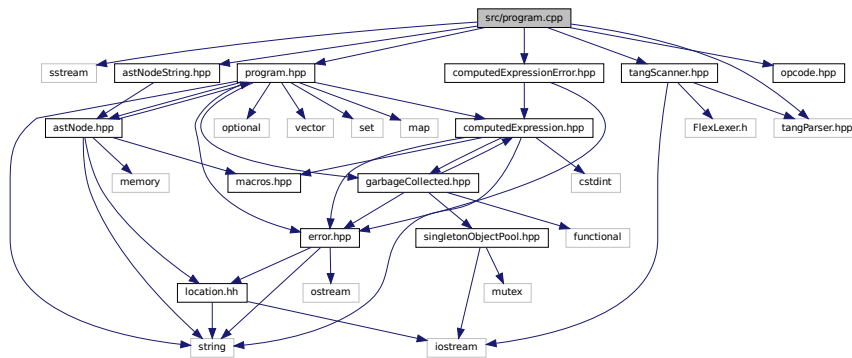
Define the [Tang::Program](#) class.

```

#include <sstream>
#include "program.hpp"
#include "opcode.hpp"
#include "tangScanner.hpp"
#include "tangParser.hpp"
#include "astNodeString.hpp"
#include "computedExpressionError.hpp"

```

Include dependency graph for program.cpp:



6.83.1 Detailed Description

Define the [Tang::Program](#) class.

6.84 src/tangBase.cpp File Reference

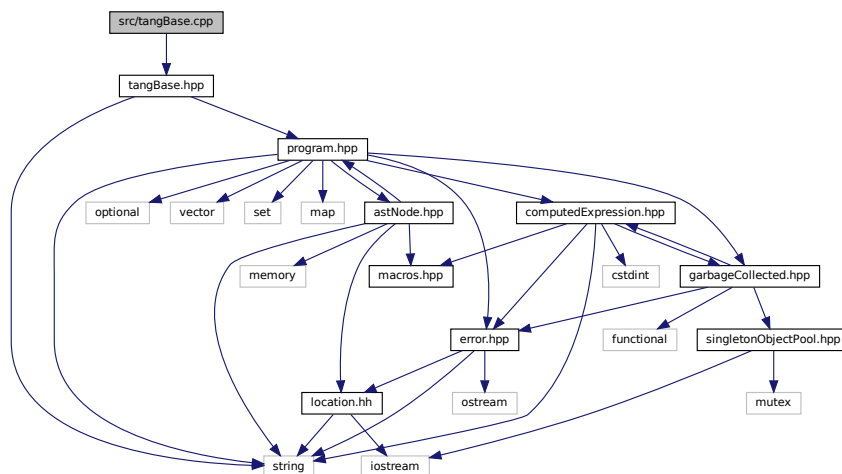
Define the [Tang::TangBase](#) class.

```

#include "tangBase.hpp"

```

Include dependency graph for tangBase.cpp:



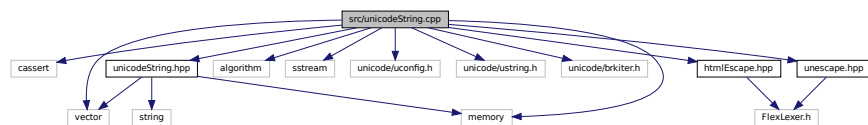
6.84.1 Detailed Description

Define the [Tang::TangBase](#) class.

6.85 src/unicodeString.cpp File Reference

Contains the function declarations for the [Tang::UnicodeString](#) class and the interface to ICU.

```
#include <cassert>
#include <vector>
#include <memory>
#include <algorithm>
#include <sstream>
#include <unicode/uconfig.h>
#include <unicode/ustring.h>
#include <unicode/brkiter.h>
#include "unicodeString.hpp"
#include "unescape.hpp"
#include "htmlEscape.hpp"
Include dependency graph for unicodeString.cpp:
```



6.85.1 Detailed Description

Contains the function declarations for the [Tang::UnicodeString](#) class and the interface to ICU.

6.86 test/test.cpp File Reference

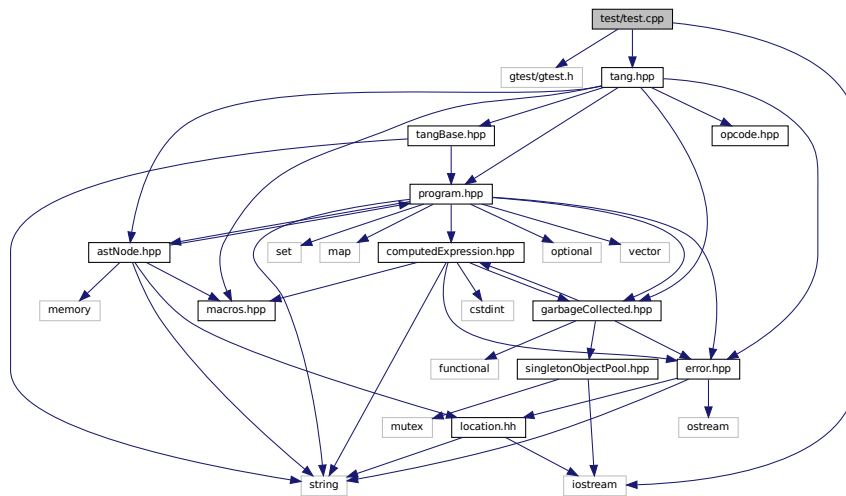
Test the general language behaviors.

```
#include <gtest/gtest.h>
#include <iostream>
```



```
#include "tang.hpp"
```

Include dependency graph for test.cpp:



Functions

- **TEST** (Declare, Null)
- **TEST** (Declare, Integer)
- **TEST** (Declare, Float)
- **TEST** (Declare, Boolean)
- **TEST** (Declare, String)
- **TEST** (Expression, Add)
- **TEST** (Expression, Subtract)
- **TEST** (Expression, Multiplication)
- **TEST** (Expression, Division)
- **TEST** (Expression, Modulo)
- **TEST** (Expression, UnaryMinus)
- **TEST** (Expression, Parentheses)
- **TEST** (Expression, TypeCast)
- **TEST** (Expression, Not)
- **TEST** (Expression, LessThan)
- **TEST** (Expression, LessThanEqual)
- **TEST** (Expression, GreaterThan)
- **TEST** (Expression, GreaterThanEqual)
- **TEST** (Expression, Equal)
- **TEST** (Expression, NotEqual)
- **TEST** (Expression, And)
- **TEST** (Expression, Or)
- **TEST** (Expression, Ternary)
- **TEST** (Expression, StringIndex)
- **TEST** (Expression, StringSlice)
- **TEST** (Expression, ArrayIndex)
- **TEST** (CodeBlock, Statements)
- **TEST** (Assign, Identifier)
- **TEST** (Assign, Index)
- **TEST** (Expression, ArraySlice)

- **TEST** (ControlFlow, IfElse)
- **TEST** (ControlFlow, While)
- **TEST** (ControlFlow, Break)
- **TEST** (ControlFlow, Continue)
- **TEST** (ControlFlow, DoWhile)
- **TEST** (ControlFlow, For)
- **TEST** (Print, Default)
- **TEST** (Print, Array)

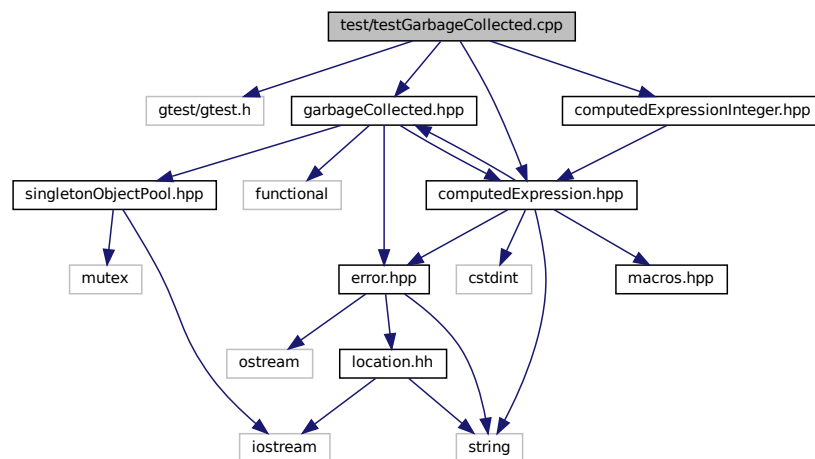
6.86.1 Detailed Description

Test the general language behaviors.

6.87 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, ObjectsRecycled)
- **TEST** (Recycle, ObjectsNotRecycled)
- `int main` (int argc, char **argv)

6.87.1 Detailed Description

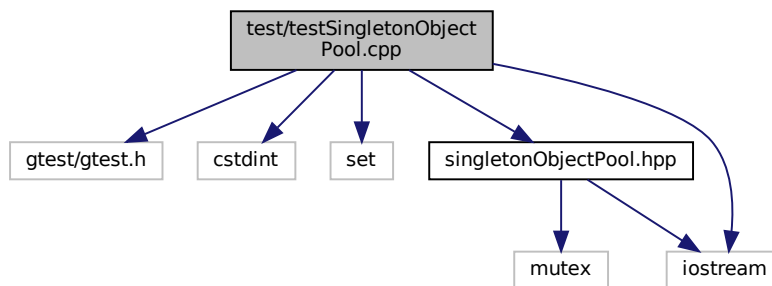
Test the generic behavior of the [Tang::GarbageCollected](#) class.

6.88 test/testSingletonObjectPool.cpp File Reference

Test the generic behavior of the [Tang::SingletonObjectPool](#) class.

```
#include <gtest/gtest.h>
#include <cstdlib>
#include <set>
#include "singletonObjectPool.hpp"
#include <iostream>
```

Include dependency graph for testSingletonObjectPool.cpp:



Functions

- **TEST** (Singleton, SameForSameType)
- **TEST** (Singleton, DifferentForDifferentTypes)
- **TEST** (Get, SuccessiveCallsProduceDifferentMemoryAddresses)
- **TEST** (Recycle, RecycledObjectIsReused)
- **TEST** (Get, SuccessiveCallsAreSequential)
- **TEST** (Get, KeepsGeneratingDifferentPointers)
- **TEST** (Recycle, WorksAfterLargeNumberOfAllocations)
- `int main` (int argc, char **argv)

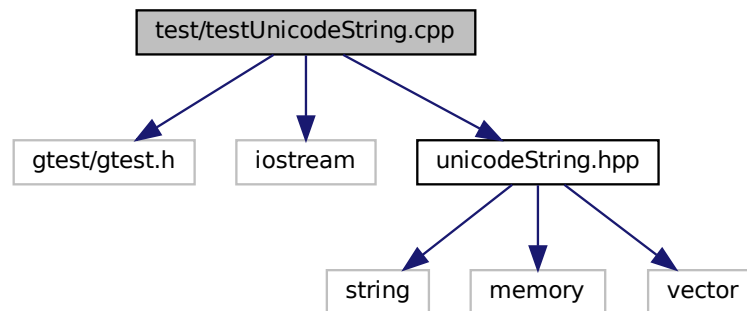
6.88.1 Detailed Description

Test the generic behavior of the [Tang::SingletonObjectPool](#) class.

6.89 test/testUnicodeString.cpp File Reference

Contains tests for the [Tang::UnicodeString](#) class.

```
#include <gtest/gtest.h>
#include <iostream>
#include "unicodeString.hpp"
Include dependency graph for testUnicodeString.cpp:
```



Functions

- **TEST** (Core, [Unescape](#))
- **TEST** (Core, [HtmlEscape](#))
- **TEST** ([UnicodeString](#), SubString)
- `int main` (int argc, char **argv)

6.89.1 Detailed Description

Contains tests for the [Tang::UnicodeString](#) class.

Index

- __add
 - Tang::ComputedExpression, 108
 - Tang::ComputedExpressionArray, 120
 - Tang::ComputedExpressionBoolean, 132
 - Tang::ComputedExpressionCompiledFunction, 144
 - Tang::ComputedExpressionError, 156
 - Tang::ComputedExpressionFloat, 168
 - Tang::ComputedExpressionInteger, 179
 - Tang::ComputedExpressionString, 191
- __asCode
 - Tang::ComputedExpression, 109
 - Tang::ComputedExpressionArray, 121
 - Tang::ComputedExpressionBoolean, 133
 - Tang::ComputedExpressionCompiledFunction, 144
 - Tang::ComputedExpressionError, 157
 - Tang::ComputedExpressionFloat, 168
 - Tang::ComputedExpressionInteger, 180
 - Tang::ComputedExpressionString, 191
- __assign_index
 - Tang::ComputedExpression, 109
 - Tang::ComputedExpressionArray, 121
 - Tang::ComputedExpressionBoolean, 133
 - Tang::ComputedExpressionCompiledFunction, 144
 - Tang::ComputedExpressionError, 157
 - Tang::ComputedExpressionFloat, 168
 - Tang::ComputedExpressionInteger, 180
 - Tang::ComputedExpressionString, 192
- __boolean
 - Tang::ComputedExpression, 109
 - Tang::ComputedExpressionArray, 121
 - Tang::ComputedExpressionBoolean, 133
 - Tang::ComputedExpressionCompiledFunction, 145
 - Tang::ComputedExpressionError, 157
 - Tang::ComputedExpressionFloat, 169
 - Tang::ComputedExpressionInteger, 180
 - Tang::ComputedExpressionString, 192
- __divide
 - Tang::ComputedExpression, 110
 - Tang::ComputedExpressionArray, 122
 - Tang::ComputedExpressionBoolean, 133
 - Tang::ComputedExpressionCompiledFunction, 145
 - Tang::ComputedExpressionError, 157
 - Tang::ComputedExpressionFloat, 169
 - Tang::ComputedExpressionInteger, 181
 - Tang::ComputedExpressionString, 193
- __equal
 - Tang::ComputedExpression, 110
 - Tang::ComputedExpressionArray, 122
 - Tang::ComputedExpressionBoolean, 134
 - Tang::ComputedExpressionCompiledFunction, 145
 - Tang::ComputedExpressionError, 158
 - Tang::ComputedExpressionFloat, 169
 - Tang::ComputedExpressionInteger, 181
 - Tang::ComputedExpressionString, 193
- __float
 - Tang::ComputedExpression, 111
 - Tang::ComputedExpressionArray, 122
 - Tang::ComputedExpressionBoolean, 134
 - Tang::ComputedExpressionCompiledFunction, 146
 - Tang::ComputedExpressionError, 158
 - Tang::ComputedExpressionFloat, 170
 - Tang::ComputedExpressionInteger, 181
 - Tang::ComputedExpressionString, 194
- __index
 - Tang::ComputedExpression, 111
 - Tang::ComputedExpressionArray, 123
 - Tang::ComputedExpressionBoolean, 134
 - Tang::ComputedExpressionCompiledFunction, 146
 - Tang::ComputedExpressionError, 158
 - Tang::ComputedExpressionFloat, 170
 - Tang::ComputedExpressionInteger, 182
 - Tang::ComputedExpressionString, 194
- __integer
 - Tang::ComputedExpression, 111
 - Tang::ComputedExpressionArray, 123
 - Tang::ComputedExpressionBoolean, 135
 - Tang::ComputedExpressionCompiledFunction, 146
 - Tang::ComputedExpressionError, 159
 - Tang::ComputedExpressionFloat, 170
 - Tang::ComputedExpressionInteger, 182
 - Tang::ComputedExpressionString, 194
- __lessThan
 - Tang::ComputedExpression, 111
 - Tang::ComputedExpressionArray, 123
 - Tang::ComputedExpressionBoolean, 135
 - Tang::ComputedExpressionCompiledFunction, 147
 - Tang::ComputedExpressionError, 159
 - Tang::ComputedExpressionFloat, 171
 - Tang::ComputedExpressionInteger, 182
 - Tang::ComputedExpressionString, 195
- __modulo
 - Tang::ComputedExpression, 112
 - Tang::ComputedExpressionArray, 124
 - Tang::ComputedExpressionBoolean, 136
 - Tang::ComputedExpressionCompiledFunction, 147
 - Tang::ComputedExpressionError, 159
 - Tang::ComputedExpressionFloat, 171
 - Tang::ComputedExpressionInteger, 183

- Tang::ComputedExpressionString, 195
- __multiply
 - Tang::ComputedExpression, 112
 - Tang::ComputedExpressionArray, 124
 - Tang::ComputedExpressionBoolean, 136
 - Tang::ComputedExpressionCompiledFunction, 148
 - Tang::ComputedExpressionError, 160
 - Tang::ComputedExpressionFloat, 171
 - Tang::ComputedExpressionInteger, 183
 - Tang::ComputedExpressionString, 196
- __negative
 - Tang::ComputedExpression, 113
 - Tang::ComputedExpressionArray, 124
 - Tang::ComputedExpressionBoolean, 136
 - Tang::ComputedExpressionCompiledFunction, 148
 - Tang::ComputedExpressionError, 160
 - Tang::ComputedExpressionFloat, 172
 - Tang::ComputedExpressionInteger, 183
 - Tang::ComputedExpressionString, 196
- __not
 - Tang::ComputedExpression, 113
 - Tang::ComputedExpressionArray, 125
 - Tang::ComputedExpressionBoolean, 136
 - Tang::ComputedExpressionCompiledFunction, 148
 - Tang::ComputedExpressionError, 160
 - Tang::ComputedExpressionFloat, 172
 - Tang::ComputedExpressionInteger, 184
 - Tang::ComputedExpressionString, 196
- __slice
 - Tang::ComputedExpression, 113
 - Tang::ComputedExpressionArray, 125
 - Tang::ComputedExpressionBoolean, 137
 - Tang::ComputedExpressionCompiledFunction, 148
 - Tang::ComputedExpressionError, 161
 - Tang::ComputedExpressionFloat, 172
 - Tang::ComputedExpressionInteger, 184
 - Tang::ComputedExpressionString, 197
- __string
 - Tang::ComputedExpression, 114
 - Tang::ComputedExpressionArray, 126
 - Tang::ComputedExpressionBoolean, 137
 - Tang::ComputedExpressionCompiledFunction, 149
 - Tang::ComputedExpressionError, 161
 - Tang::ComputedExpressionFloat, 173
 - Tang::ComputedExpressionInteger, 185
 - Tang::ComputedExpressionString, 198
- __subtract
 - Tang::ComputedExpression, 114
 - Tang::ComputedExpressionArray, 126
 - Tang::ComputedExpressionBoolean, 137
 - Tang::ComputedExpressionCompiledFunction, 149
 - Tang::ComputedExpressionError, 161
 - Tang::ComputedExpressionFloat, 173
 - Tang::ComputedExpressionInteger, 185
 - Tang::ComputedExpressionString, 198
- ~GarbageCollected
 - Tang::GarbageCollected, 207
- ADD
 - opcode.hpp, 284
- Add
 - Tang::AstNodeBinary, 25
- addBreak
 - Tang::Program, 228
- addBytecode
 - Tang::Program, 228
- addContinue
 - Tang::Program, 228
- addIdentifier
 - Tang::Program, 228
- addIdentifierAssigned
 - Tang::Program, 229
- addString
 - Tang::Program, 229
- And
 - Tang::AstNodeBinary, 25
- ARRAY
 - opcode.hpp, 284
- ASSIGNINDEX
 - opcode.hpp, 284
- AstNode
 - Tang::AstNode, 13
- AstNodeArray
 - Tang::AstNodeArray, 17
- AstNodeAssign
 - Tang::AstNodeAssign, 21
- AstNodeBinary
 - Tang::AstNodeBinary, 25
- AstNodeBlock
 - Tang::AstNodeBlock, 29
- AstNodeBoolean
 - Tang::AstNodeBoolean, 32
- AstNodeBreak
 - Tang::AstNodeBreak, 36
- AstNodeCast
 - Tang::AstNodeCast, 40
- AstNodeContinue
 - Tang::AstNodeContinue, 44
- AstNodeDoWhile
 - Tang::AstNodeDoWhile, 47
- AstNodeFloat
 - Tang::AstNodeFloat, 51
- AstNodeFor
 - Tang::AstNodeFor, 54
- AstNodeFunctionCall
 - Tang::AstNodeFunctionCall, 58
- AstNodeFunctionDeclaration
 - Tang::AstNodeFunctionDeclaration, 61
- AstNodeIdentifier
 - Tang::AstNodeIdentifier, 65
- AstNodeIfElse
 - Tang::AstNodeIfElse, 69, 70
- AstNodeIndex
 - Tang::AstNodeIndex, 73
- AstNodeInteger
 - Tang::AstNodeInteger, 77
- AstNodePrint

- Tang::AstNodePrint, 81
- AstNodeReturn
 - Tang::AstNodeReturn, 84
- AstNodeSlice
 - Tang::AstNodeSlice, 88
- AstNodeString
 - Tang::AstNodeString, 92
- AstNodeTernary
 - Tang::AstNodeTernary, 97
- AstNodeUnary
 - Tang::AstNodeUnary, 101
- AstNodeWhile
 - Tang::AstNodeWhile, 104
- BOOLEAN
 - opcode.hpp, 284
- Boolean
 - Tang::AstNodeCast, 40
- build/generated/location.hh, 245
- bytesLength
 - Tang::UnicodeString, 242
- CALLFUNC
 - opcode.hpp, 285
- CASTBOOLEAN
 - opcode.hpp, 285
- CASTFLOAT
 - opcode.hpp, 285
- CASTINTEGER
 - opcode.hpp, 284
- CodeType
 - Tang::Program, 227
- compile
 - Tang::AstNode, 14
 - Tang::AstNodeArray, 17
 - Tang::AstNodeAssign, 22
 - Tang::AstNodeBinary, 26
 - Tang::AstNodeBlock, 29
 - Tang::AstNodeBoolean, 32
 - Tang::AstNodeBreak, 37
 - Tang::AstNodeCast, 41
 - Tang::AstNodeContinue, 44
 - Tang::AstNodeDoWhile, 48
 - Tang::AstNodeFloat, 51
 - Tang::AstNodeFor, 55
 - Tang::AstNodeFunctionCall, 58
 - Tang::AstNodeFunctionDeclaration, 62
 - Tang::AstNodeIdentifier, 66
 - Tang::AstNodeIfElse, 70
 - Tang::AstNodeIndex, 73
 - Tang::AstNodeInteger, 77
 - Tang::AstNodePrint, 81
 - Tang::AstNodeReturn, 84
 - Tang::AstNodeSlice, 89
 - Tang::AstNodeString, 92
 - Tang::AstNodeTernary, 97
 - Tang::AstNodeUnary, 101
 - Tang::AstNodeWhile, 104
- compileLiteral
 - Tang::AstNodeString, 94
- compilePreprocess
 - Tang::AstNode, 14
 - Tang::AstNodeArray, 19
 - Tang::AstNodeAssign, 22
 - Tang::AstNodeBinary, 26
 - Tang::AstNodeBlock, 30
 - Tang::AstNodeBoolean, 34
 - Tang::AstNodeBreak, 37
 - Tang::AstNodeCast, 41
 - Tang::AstNodeContinue, 45
 - Tang::AstNodeDoWhile, 48
 - Tang::AstNodeFloat, 52
 - Tang::AstNodeFor, 55
 - Tang::AstNodeFunctionCall, 59
 - Tang::AstNodeFunctionDeclaration, 63
 - Tang::AstNodeIdentifier, 66
 - Tang::AstNodeIfElse, 71
 - Tang::AstNodeIndex, 74
 - Tang::AstNodeInteger, 78
 - Tang::AstNodePrint, 82
 - Tang::AstNodeReturn, 86
 - Tang::AstNodeSlice, 89
 - Tang::AstNodeString, 94
 - Tang::AstNodeTernary, 98
 - Tang::AstNodeUnary, 102
 - Tang::AstNodeWhile, 105
- compileScript
 - Tang::TangBase, 236
- ComputedExpressionArray
 - Tang::ComputedExpressionArray, 120
- ComputedExpressionBoolean
 - Tang::ComputedExpressionBoolean, 132
- ComputedExpressionCompiledFunction
 - Tang::ComputedExpressionCompiledFunction, 143
- ComputedExpressionError
 - Tang::ComputedExpressionError, 156
- ComputedExpressionFloat
 - Tang::ComputedExpressionFloat, 167
- ComputedExpressionInteger
 - Tang::ComputedExpressionInteger, 179
- ComputedExpressionString
 - Tang::ComputedExpressionString, 191
- COPY
 - opcode.hpp, 284
- Default
 - Tang::AstNode, 13
 - Tang::AstNodeArray, 17
 - Tang::AstNodeAssign, 21
 - Tang::AstNodeBinary, 25
 - Tang::AstNodeBlock, 29
 - Tang::AstNodeBoolean, 32
 - Tang::AstNodeBreak, 36
 - Tang::AstNodeCast, 40
 - Tang::AstNodeContinue, 44
 - Tang::AstNodeDoWhile, 47
 - Tang::AstNodeFloat, 51
 - Tang::AstNodeFor, 54

- Tang::AstNodeFunctionCall, [58](#)
- Tang::AstNodeFunctionDeclaration, [61](#)
- Tang::AstNodeIdentifier, [65](#)
- Tang::AstNodeIfElse, [69](#)
- Tang::AstNodeIndex, [73](#)
- Tang::AstNodeInteger, [77](#)
- Tang::AstNodePrint, [80](#), [81](#)
- Tang::AstNodeReturn, [84](#)
- Tang::AstNodeSlice, [88](#)
- Tang::AstNodeString, [92](#)
- Tang::AstNodeTernary, [97](#)
- Tang::AstNodeUnary, [101](#)
- Tang::AstNodeWhile, [104](#)
- DIVIDE
 - opcode.hpp, [284](#)
- Divide
 - Tang::AstNodeBinary, [25](#)
- dump
 - Tang::AstNode, [15](#)
 - Tang::AstNodeArray, [19](#)
 - Tang::AstNodeAssign, [23](#)
 - Tang::AstNodeBinary, [27](#)
 - Tang::AstNodeBlock, [30](#)
 - Tang::AstNodeBoolean, [34](#)
 - Tang::AstNodeBreak, [38](#)
 - Tang::AstNodeCast, [42](#)
 - Tang::AstNodeContinue, [45](#)
 - Tang::AstNodeDoWhile, [49](#)
 - Tang::AstNodeFloat, [52](#)
 - Tang::AstNodeFor, [56](#)
 - Tang::AstNodeFunctionCall, [59](#)
 - Tang::AstNodeFunctionDeclaration, [63](#)
 - Tang::AstNodeIdentifier, [67](#)
 - Tang::AstNodeIfElse, [71](#)
 - Tang::AstNodeIndex, [74](#)
 - Tang::AstNodeInteger, [78](#)
 - Tang::AstNodePrint, [82](#)
 - Tang::AstNodeReturn, [86](#)
 - Tang::AstNodeSlice, [90](#)
 - Tang::AstNodeString, [95](#)
 - Tang::AstNodeTernary, [98](#)
 - Tang::AstNodeUnary, [102](#)
 - Tang::AstNodeWhile, [105](#)
 - Tang::ComputedExpression, [114](#)
 - Tang::ComputedExpressionArray, [126](#)
 - Tang::ComputedExpressionBoolean, [138](#)
 - Tang::ComputedExpressionCompiledFunction, [150](#)
 - Tang::ComputedExpressionError, [162](#)
 - Tang::ComputedExpressionFloat, [174](#)
 - Tang::ComputedExpressionInteger, [185](#)
 - Tang::ComputedExpressionString, [198](#)
- dumpBytecode
 - Tang::Program, [229](#)
- DUMPPROGRAMCHECK
 - program-dumpBytecode.cpp, [318](#)
- EQ
 - opcode.hpp, [284](#)
- Equal
 - Tang::AstNodeBinary, [25](#)
- Error
 - Tang::Error, [203](#)
- error.cpp
 - operator<<, [317](#)
- execute
 - Tang::Program, [230](#)
- EXECUTEPROGRAMCHECK
 - program-execute.cpp, [320](#)
- FLOAT
 - opcode.hpp, [284](#)
- Float
 - Tang::AstNodeCast, [40](#)
- FUNCTION
 - opcode.hpp, [284](#)
- functionsDeclared
 - Tang::Program, [234](#)
- GarbageCollected
 - Tang::GarbageCollected, [206](#), [207](#)
- get
 - Tang::SingletonObjectPool< T >, [235](#)
- get_next_token
 - Tang::HtmlEscape, [222](#)
 - Tang::TangScanner, [238](#)
 - Tang::Unescape, [240](#)
- getAst
 - Tang::Program, [230](#)
- getBytecode
 - Tang::Program, [230](#)
- getCode
 - Tang::Program, [230](#)
- getCollection
 - Tang::AstNodeIndex, [75](#)
- getIdentifiers
 - Tang::Program, [231](#)
- getIdentifiersAssigned
 - Tang::Program, [231](#)
- getIndex
 - Tang::AstNodeIndex, [75](#)
- getInstance
 - Tang::SingletonObjectPool< T >, [235](#)
- getResult
 - Tang::Program, [231](#)
- getStrings
 - Tang::Program, [231](#)
- GreaterThan
 - Tang::AstNodeBinary, [25](#)
- GreaterThanEqual
 - Tang::AstNodeBinary, [25](#)
- GT
 - opcode.hpp, [284](#)
- GTE
 - opcode.hpp, [284](#)
- HtmlEscape
 - Tang::HtmlEscape, [221](#)
- htmlEscape

- unicodeString.hpp, 292
- include/astNode.hpp, 247
- include/astNodeArray.hpp, 248
- include/astNodeAssign.hpp, 249
- include/astNodeBinary.hpp, 250
- include/astNodeBlock.hpp, 251
- include/astNodeBoolean.hpp, 252
- include/astNodeBreak.hpp, 253
- include/astNodeCast.hpp, 254
- include/astNodeContinue.hpp, 255
- include/astNodeDoWhile.hpp, 256
- include/astNodeFloat.hpp, 257
- include/astNodeFor.hpp, 258
- include/astNodeFunctionCall.hpp, 259
- include/astNodeFunctionDeclaration.hpp, 260
- include/astNodeIdentifier.hpp, 261
- include/astNodeIfElse.hpp, 262
- include/astNodeIndex.hpp, 263
- include/astNodeInteger.hpp, 264
- include/astNodePrint.hpp, 265
- include/astNodeReturn.hpp, 266
- include/astNodeSlice.hpp, 267
- include/astNodeString.hpp, 268
- include/astNodeTernary.hpp, 269
- include/astNodeUnary.hpp, 270
- include/astNodeWhile.hpp, 271
- include/computedExpression.hpp, 272
- include/computedExpressionArray.hpp, 273
- include/computedExpressionBoolean.hpp, 274
- include/computedExpressionCompiledFunction.hpp, 275
- include/computedExpressionError.hpp, 276
- include/computedExpressionFloat.hpp, 277
- include/computedExpressionInteger.hpp, 278
- include/computedExpressionString.hpp, 279
- include/error.hpp, 279
- include/garbageCollected.hpp, 280
- include/htmlEscape.hpp, 281
- include/macros.hpp, 283
- include/opcode.hpp, 283
- include/program.hpp, 285
- include/singletonObjectPool.hpp, 286
- include/tang.hpp, 287
- include/tangBase.hpp, 288
- include/tangScanner.hpp, 289
- include/unescape.hpp, 290
- include/unicodeString.hpp, 291
- INDEX
 - opcode.hpp, 284
- INTEGER
 - opcode.hpp, 284
- Integer
 - Tang::AstNodeCast, 40
- is_equal
 - Tang::ComputedExpression, 115–117
 - Tang::ComputedExpressionArray, 127–129
 - Tang::ComputedExpressionBoolean, 138–140
 - Tang::ComputedExpressionCompiledFunction, 150, 152, 153
 - Tang::ComputedExpressionError, 162–164
 - Tang::ComputedExpressionFloat, 174–176
 - Tang::ComputedExpressionInteger, 186–188
 - Tang::ComputedExpressionString, 198–200
- IsAssignment
 - Tang::AstNode, 13
 - Tang::AstNodeArray, 17
 - Tang::AstNodeAssign, 21
 - Tang::AstNodeBinary, 25
 - Tang::AstNodeBlock, 29
 - Tang::AstNodeBoolean, 32
 - Tang::AstNodeBreak, 36
 - Tang::AstNodeCast, 40
 - Tang::AstNodeContinue, 44
 - Tang::AstNodeDoWhile, 47
 - Tang::AstNodeFloat, 51
 - Tang::AstNodeFor, 54
 - Tang::AstNodeFunctionCall, 58
 - Tang::AstNodeFunctionDeclaration, 61
 - Tang::AstNodeIdentifier, 65
 - Tang::AstNodeIfElse, 69
 - Tang::AstNodeIndex, 73
 - Tang::AstNodeInteger, 77
 - Tang::AstNodePrint, 80
 - Tang::AstNodeReturn, 84
 - Tang::AstNodeSlice, 88
 - Tang::AstNodeString, 92
 - Tang::AstNodeTernary, 97
 - Tang::AstNodeUnary, 101
 - Tang::AstNodeWhile, 104
- isCopyNeeded
 - Tang::ComputedExpression, 117
 - Tang::ComputedExpressionArray, 129
 - Tang::ComputedExpressionBoolean, 140
 - Tang::ComputedExpressionCompiledFunction, 153
 - Tang::ComputedExpressionError, 164
 - Tang::ComputedExpressionFloat, 176
 - Tang::ComputedExpressionInteger, 188
 - Tang::ComputedExpressionString, 201
 - Tang::GarbageCollected, 207
- JMP
 - opcode.hpp, 284
- JMPF
 - opcode.hpp, 284
- JMPF_POP
 - opcode.hpp, 284
- JMPT
 - opcode.hpp, 284
- JMPT_POP
 - opcode.hpp, 284
- length
 - Tang::UnicodeString, 242
- LessThan
 - Tang::AstNodeBinary, 25
- LessThanEqual

- Tang::AstNodeBinary, 25
- location.hh
 - operator<, 246, 247
- LT
 - opcode.hpp, 284
- LTE
 - opcode.hpp, 284
- make
 - Tang::GarbageCollected, 208
- makeCopy
 - Tang::ComputedExpression, 117
 - Tang::ComputedExpressionArray, 129
 - Tang::ComputedExpressionBoolean, 141
 - Tang::ComputedExpressionCompiledFunction, 153
 - Tang::ComputedExpressionError, 165
 - Tang::ComputedExpressionFloat, 176
 - Tang::ComputedExpressionInteger, 188
 - Tang::ComputedExpressionString, 201
 - Tang::GarbageCollected, 208
- MODULO
 - opcode.hpp, 284
- Modulo
 - Tang::AstNodeBinary, 25
- MULTIPLY
 - opcode.hpp, 284
- Multiply
 - Tang::AstNodeBinary, 25
- NEGATIVE
 - opcode.hpp, 284
- Negative
 - Tang::AstNodeUnary, 100
- NEQ
 - opcode.hpp, 284
- NOT
 - opcode.hpp, 284
- Not
 - Tang::AstNodeUnary, 100
- NotEqual
 - Tang::AstNodeBinary, 25
- NULLVAL
 - opcode.hpp, 284
- Opcode
 - opcode.hpp, 284
- opcode.hpp
 - ADD, 284
 - ARRAY, 284
 - ASSIGNINDEX, 284
 - BOOLEAN, 284
 - CALLFUNC, 285
 - CASTBOOLEAN, 285
 - CASTFLOAT, 285
 - CASTINTEGER, 284
 - COPY, 284
 - DIVIDE, 284
 - EQ, 284
 - FLOAT, 284
 - FUNCTION, 284
 - GT, 284
 - GTE, 284
 - INDEX, 284
 - INTEGER, 284
 - JMP, 284
 - JMPF, 284
 - JMPF_POP, 284
 - JMPT, 284
 - JMPT_POP, 284
 - LT, 284
 - LTE, 284
 - MODULO, 284
 - MULTIPLY, 284
 - NEGATIVE, 284
 - NEQ, 284
 - NOT, 284
 - NULLVAL, 284
 - Opcode, 284
 - PEEK, 284
 - POKE, 284
 - POP, 284
 - PRINT, 285
 - RETURN, 285
 - SLICE, 284
 - STRING, 284
 - SUBTRACT, 284
- Operation
 - Tang::AstNodeBinary, 24
- Operator
 - Tang::AstNodeUnary, 100
- operator std::string
 - Tang::UnicodeString, 242
- operator!
 - Tang::GarbageCollected, 209
- operator!=
 - Tang::GarbageCollected, 209
- operator<
 - Tang::GarbageCollected, 214
 - Tang::UnicodeString, 243
- operator<<
 - error.cpp, 317
 - location.hh, 246, 247
 - Tang::Error, 203
 - Tang::GarbageCollected, 219
- operator<=
 - Tang::GarbageCollected, 214
- operator>
 - Tang::GarbageCollected, 218
- operator>=
 - Tang::GarbageCollected, 219
- operator*
 - Tang::GarbageCollected, 210, 211
- operator+
 - Tang::GarbageCollected, 211
 - Tang::UnicodeString, 242
- operator-
 - Tang::GarbageCollected, 212

- operator->
 - Tang::GarbageCollected, [213](#)
- operator/
 - Tang::GarbageCollected, [213](#)
- operator=
 - Tang::GarbageCollected, [215](#)
- operator==
 - Tang::GarbageCollected, [215–218](#)
 - Tang::UnicodeString, [243](#)
- operator%
 - Tang::GarbageCollected, [210](#)
- Or
 - Tang::AstNodeBinary, [25](#)
- PEEK
 - opcode.hpp, [284](#)
- POKE
 - opcode.hpp, [284](#)
- POP
 - opcode.hpp, [284](#)
- popBreakStack
 - Tang::Program, [232](#)
- popContinueStack
 - Tang::Program, [232](#)
- PreprocessState
 - Tang::AstNode, [13](#)
 - Tang::AstNodeArray, [17](#)
 - Tang::AstNodeAssign, [21](#)
 - Tang::AstNodeBinary, [25](#)
 - Tang::AstNodeBlock, [28](#)
 - Tang::AstNodeBoolean, [32](#)
 - Tang::AstNodeBreak, [36](#)
 - Tang::AstNodeCast, [40](#)
 - Tang::AstNodeContinue, [43](#)
 - Tang::AstNodeDoWhile, [47](#)
 - Tang::AstNodeFloat, [50](#)
 - Tang::AstNodeFor, [54](#)
 - Tang::AstNodeFunctionCall, [58](#)
 - Tang::AstNodeFunctionDeclaration, [61](#)
 - Tang::AstNodeIdentifier, [65](#)
 - Tang::AstNodeIfElse, [69](#)
 - Tang::AstNodeIndex, [73](#)
 - Tang::AstNodeInteger, [77](#)
 - Tang::AstNodePrint, [80](#)
 - Tang::AstNodeReturn, [84](#)
 - Tang::AstNodeSlice, [88](#)
 - Tang::AstNodeString, [92](#)
 - Tang::AstNodeTernary, [97](#)
 - Tang::AstNodeUnary, [100](#)
 - Tang::AstNodeWhile, [104](#)
- PRINT
 - opcode.hpp, [285](#)
- Program
 - Tang::Program, [227](#)
- program-dumpBytecode.cpp
 - DUMPPROGRAMCHECK, [318](#)
- program-execute.cpp
 - EXECUTEPROGRAMCHECK, [320](#)
 - STACKCHECK, [320](#)
- pushEnvironment
 - Tang::Program, [233](#)
- recycle
 - Tang::SingletonObjectPool< T >, [235](#)
- RETURN
 - opcode.hpp, [285](#)
- Script
 - Tang::Program, [227](#)
- setFunctionStackDeclaration
 - Tang::Program, [233](#)
- setJumpTarget
 - Tang::Program, [234](#)
- SLICE
 - opcode.hpp, [284](#)
- src/astNode.cpp, [293](#)
- src/astNodeArray.cpp, [294](#)
- src/astNodeAssign.cpp, [294](#)
- src/astNodeBinary.cpp, [295](#)
- src/astNodeBlock.cpp, [296](#)
- src/astNodeBoolean.cpp, [296](#)
- src/astNodeBreak.cpp, [297](#)
- src/astNodeCast.cpp, [298](#)
- src/astNodeContinue.cpp, [298](#)
- src/astNodeDoWhile.cpp, [299](#)
- src/astNodeFloat.cpp, [300](#)
- src/astNodeFor.cpp, [301](#)
- src/astNodeFunctionCall.cpp, [301](#)
- src/astNodeFunctionDeclaration.cpp, [302](#)
- src/astNodeIdentifier.cpp, [303](#)
- src/astNodeIfElse.cpp, [304](#)
- src/astNodeIndex.cpp, [304](#)
- src/astNodeInteger.cpp, [305](#)
- src/astNodePrint.cpp, [306](#)
- src/astNodeReturn.cpp, [306](#)
- src/astNodeSlice.cpp, [307](#)
- src/astNodeString.cpp, [308](#)
- src/astNodeTernary.cpp, [309](#)
- src/astNodeUnary.cpp, [310](#)
- src/astNodeWhile.cpp, [310](#)
- src/computedExpression.cpp, [311](#)
- src/computedExpressionArray.cpp, [312](#)
- src/computedExpressionBoolean.cpp, [313](#)
- src/computedExpressionCompiledFunction.cpp, [313](#)
- src/computedExpressionError.cpp, [314](#)
- src/computedExpressionFloat.cpp, [315](#)
- src/computedExpressionInteger.cpp, [315](#)
- src/computedExpressionString.cpp, [316](#)
- src/error.cpp, [317](#)
- src/program-dumpBytecode.cpp, [318](#)
- src/program-execute.cpp, [319](#)
- src/program.cpp, [320](#)
- src/tangBase.cpp, [321](#)
- src/unicodeString.cpp, [322](#)
- STACKCHECK
 - program-execute.cpp, [320](#)
- STRING
 - opcode.hpp, [284](#)

- substr
 - Tang::UnicodeString, 243
- SUBTRACT
 - opcode.hpp, 284
- Subtract
 - Tang::AstNodeBinary, 25
- Tang::AstNode, 11
 - AstNode, 13
 - compile, 14
 - compilePreprocess, 14
 - Default, 13
 - dump, 15
 - IsAssignment, 13
 - PreprocessState, 13
- Tang::AstNodeArray, 16
 - AstNodeArray, 17
 - compile, 17
 - compilePreprocess, 19
 - Default, 17
 - dump, 19
 - IsAssignment, 17
 - PreprocessState, 17
- Tang::AstNodeAssign, 20
 - AstNodeAssign, 21
 - compile, 22
 - compilePreprocess, 22
 - Default, 21
 - dump, 23
 - IsAssignment, 21
 - PreprocessState, 21
- Tang::AstNodeBinary, 23
 - Add, 25
 - And, 25
 - AstNodeBinary, 25
 - compile, 26
 - compilePreprocess, 26
 - Default, 25
 - Divide, 25
 - dump, 27
 - Equal, 25
 - GreaterThan, 25
 - GreaterThanEqual, 25
 - IsAssignment, 25
 - LessThan, 25
 - LessThanEqual, 25
 - Modulo, 25
 - Multiply, 25
 - NotEqual, 25
 - Operation, 24
 - Or, 25
 - PreprocessState, 25
 - Subtract, 25
- Tang::AstNodeBlock, 27
 - AstNodeBlock, 29
 - compile, 29
 - compilePreprocess, 30
 - Default, 29
 - dump, 30
 - IsAssignment, 29
 - PreprocessState, 28
- Tang::AstNodeBoolean, 31
 - AstNodeBoolean, 32
 - compile, 32
 - compilePreprocess, 34
 - Default, 32
 - dump, 34
 - IsAssignment, 32
 - PreprocessState, 32
- Tang::AstNodeBreak, 35
 - AstNodeBreak, 36
 - compile, 37
 - compilePreprocess, 37
 - Default, 36
 - dump, 38
 - IsAssignment, 36
 - PreprocessState, 36
- Tang::AstNodeCast, 38
 - AstNodeCast, 40
 - Boolean, 40
 - compile, 41
 - compilePreprocess, 41
 - Default, 40
 - dump, 42
 - Float, 40
 - Integer, 40
 - IsAssignment, 40
 - PreprocessState, 40
 - Type, 40
- Tang::AstNodeContinue, 42
 - AstNodeContinue, 44
 - compile, 44
 - compilePreprocess, 45
 - Default, 44
 - dump, 45
 - IsAssignment, 44
 - PreprocessState, 43
- Tang::AstNodeDoWhile, 46
 - AstNodeDoWhile, 47
 - compile, 48
 - compilePreprocess, 48
 - Default, 47
 - dump, 49
 - IsAssignment, 47
 - PreprocessState, 47
- Tang::AstNodeFloat, 49
 - AstNodeFloat, 51
 - compile, 51
 - compilePreprocess, 52
 - Default, 51
 - dump, 52
 - IsAssignment, 51
 - PreprocessState, 50
- Tang::AstNodeFor, 53
 - AstNodeFor, 54
 - compile, 55
 - compilePreprocess, 55

- Default, [54](#)
- dump, [56](#)
- IsAssignment, [54](#)
- PreprocessState, [54](#)
- Tang::AstNodeFunctionCall, [56](#)
 - AstNodeFunctionCall, [58](#)
 - compile, [58](#)
 - compilePreprocess, [59](#)
 - Default, [58](#)
 - dump, [59](#)
 - IsAssignment, [58](#)
 - PreprocessState, [58](#)
- Tang::AstNodeFunctionDeclaration, [60](#)
 - AstNodeFunctionDeclaration, [61](#)
 - compile, [62](#)
 - compilePreprocess, [63](#)
 - Default, [61](#)
 - dump, [63](#)
 - IsAssignment, [61](#)
 - PreprocessState, [61](#)
- Tang::AstNodeIdentifier, [64](#)
 - AstNodeIdentifier, [65](#)
 - compile, [66](#)
 - compilePreprocess, [66](#)
 - Default, [65](#)
 - dump, [67](#)
 - IsAssignment, [65](#)
 - PreprocessState, [65](#)
- Tang::AstNodeIfElse, [68](#)
 - AstNodeIfElse, [69, 70](#)
 - compile, [70](#)
 - compilePreprocess, [71](#)
 - Default, [69](#)
 - dump, [71](#)
 - IsAssignment, [69](#)
 - PreprocessState, [69](#)
- Tang::AstNodeIndex, [71](#)
 - AstNodeIndex, [73](#)
 - compile, [73](#)
 - compilePreprocess, [74](#)
 - Default, [73](#)
 - dump, [74](#)
 - getCollection, [75](#)
 - getIndex, [75](#)
 - IsAssignment, [73](#)
 - PreprocessState, [73](#)
- Tang::AstNodeInteger, [76](#)
 - AstNodeInteger, [77](#)
 - compile, [77](#)
 - compilePreprocess, [78](#)
 - Default, [77](#)
 - dump, [78](#)
 - IsAssignment, [77](#)
 - PreprocessState, [77](#)
- Tang::AstNodePrint, [79](#)
 - AstNodePrint, [81](#)
 - compile, [81](#)
 - compilePreprocess, [82](#)
- Default, [80, 81](#)
- dump, [82](#)
- IsAssignment, [80](#)
- PreprocessState, [80](#)
- Type, [80](#)
- Tang::AstNodeReturn, [83](#)
 - AstNodeReturn, [84](#)
 - compile, [84](#)
 - compilePreprocess, [86](#)
 - Default, [84](#)
 - dump, [86](#)
 - IsAssignment, [84](#)
 - PreprocessState, [84](#)
- Tang::AstNodeSlice, [87](#)
 - AstNodeSlice, [88](#)
 - compile, [89](#)
 - compilePreprocess, [89](#)
 - Default, [88](#)
 - dump, [90](#)
 - IsAssignment, [88](#)
 - PreprocessState, [88](#)
- Tang::AstNodeString, [90](#)
 - AstNodeString, [92](#)
 - compile, [92](#)
 - compileLiteral, [94](#)
 - compilePreprocess, [94](#)
 - Default, [92](#)
 - dump, [95](#)
 - IsAssignment, [92](#)
 - PreprocessState, [92](#)
- Tang::AstNodeTernary, [96](#)
 - AstNodeTernary, [97](#)
 - compile, [97](#)
 - compilePreprocess, [98](#)
 - Default, [97](#)
 - dump, [98](#)
 - IsAssignment, [97](#)
 - PreprocessState, [97](#)
- Tang::AstNodeUnary, [99](#)
 - AstNodeUnary, [101](#)
 - compile, [101](#)
 - compilePreprocess, [102](#)
 - Default, [101](#)
 - dump, [102](#)
 - IsAssignment, [101](#)
 - Negative, [100](#)
 - Not, [100](#)
 - Operator, [100](#)
 - PreprocessState, [100](#)
- Tang::AstNodeWhile, [103](#)
 - AstNodeWhile, [104](#)
 - compile, [104](#)
 - compilePreprocess, [105](#)
 - Default, [104](#)
 - dump, [105](#)
 - IsAssignment, [104](#)
 - PreprocessState, [104](#)
- Tang::ComputedExpression, [106](#)

- __add, 108
- __asCode, 109
- __assign_index, 109
- __boolean, 109
- __divide, 110
- __equal, 110
- __float, 111
- __index, 111
- __integer, 111
- __lessThan, 111
- __modulo, 112
- __multiply, 112
- __negative, 113
- __not, 113
- __slice, 113
- __string, 114
- __subtract, 114
- dump, 114
- is_equal, 115–117
- isCopyNeeded, 117
- makeCopy, 117
- Tang::ComputedExpressionArray, 118
 - __add, 120
 - __asCode, 121
 - __assign_index, 121
 - __boolean, 121
 - __divide, 122
 - __equal, 122
 - __float, 122
 - __index, 123
 - __integer, 123
 - __lessThan, 123
 - __modulo, 124
 - __multiply, 124
 - __negative, 124
 - __not, 125
 - __slice, 125
 - __string, 126
 - __subtract, 126
 - ComputedExpressionArray, 120
 - dump, 126
 - is_equal, 127–129
 - isCopyNeeded, 129
 - makeCopy, 129
- Tang::ComputedExpressionBoolean, 130
 - __add, 132
 - __asCode, 133
 - __assign_index, 133
 - __boolean, 133
 - __divide, 133
 - __equal, 134
 - __float, 134
 - __index, 134
 - __integer, 135
 - __lessThan, 135
 - __modulo, 136
 - __multiply, 136
 - __negative, 136
 - __not, 136
 - __slice, 137
 - __string, 137
 - __subtract, 137
 - ComputedExpressionBoolean, 132
 - dump, 138
 - is_equal, 138–140
 - isCopyNeeded, 140
 - makeCopy, 141
- Tang::ComputedExpressionCompiledFunction, 141
 - __add, 144
 - __asCode, 144
 - __assign_index, 144
 - __boolean, 145
 - __divide, 145
 - __equal, 145
 - __float, 146
 - __index, 146
 - __integer, 146
 - __lessThan, 147
 - __modulo, 147
 - __multiply, 148
 - __negative, 148
 - __not, 148
 - __slice, 148
 - __string, 149
 - __subtract, 149
 - ComputedExpressionCompiledFunction, 143
 - dump, 150
 - is_equal, 150, 152, 153
 - isCopyNeeded, 153
 - makeCopy, 153
- Tang::ComputedExpressionError, 154
 - __add, 156
 - __asCode, 157
 - __assign_index, 157
 - __boolean, 157
 - __divide, 157
 - __equal, 158
 - __float, 158
 - __index, 158
 - __integer, 159
 - __lessThan, 159
 - __modulo, 159
 - __multiply, 160
 - __negative, 160
 - __not, 160
 - __slice, 161
 - __string, 161
 - __subtract, 161
 - ComputedExpressionError, 156
 - dump, 162
 - is_equal, 162–164
 - isCopyNeeded, 164
 - makeCopy, 165
- Tang::ComputedExpressionFloat, 165
 - __add, 168
 - __asCode, 168

- __assign_index, 168
- __boolean, 169
- __divide, 169
- __equal, 169
- __float, 170
- __index, 170
- __integer, 170
- __lessThan, 171
- __modulo, 171
- __multiply, 171
- __negative, 172
- __not, 172
- __slice, 172
- __string, 173
- __subtract, 173
- ComputedExpressionFloat, 167
- dump, 174
- is_equal, 174–176
- isCopyNeeded, 176
- makeCopy, 176
- Tang::ComputedExpressionInteger, 177
 - __add, 179
 - __asCode, 180
 - __assign_index, 180
 - __boolean, 180
 - __divide, 181
 - __equal, 181
 - __float, 181
 - __index, 182
 - __integer, 182
 - __lessThan, 182
 - __modulo, 183
 - __multiply, 183
 - __negative, 183
 - __not, 184
 - __slice, 184
 - __string, 185
 - __subtract, 185
 - ComputedExpressionInteger, 179
 - dump, 185
 - is_equal, 186–188
 - isCopyNeeded, 188
 - makeCopy, 188
- Tang::ComputedExpressionString, 189
 - __add, 191
 - __asCode, 191
 - __assign_index, 192
 - __boolean, 192
 - __divide, 193
 - __equal, 193
 - __float, 194
 - __index, 194
 - __integer, 194
 - __lessThan, 195
 - __modulo, 195
 - __multiply, 196
 - __negative, 196
 - __not, 196
 - __slice, 197
 - __string, 198
 - __subtract, 198
 - ComputedExpressionString, 191
 - dump, 198
 - is_equal, 198–200
 - isCopyNeeded, 201
 - makeCopy, 201
- Tang::Error, 202
 - Error, 203
 - operator<<, 203
- Tang::GarbageCollected, 204
 - ~GarbageCollected, 207
 - GarbageCollected, 206, 207
 - isCopyNeeded, 207
 - make, 208
 - makeCopy, 208
 - operator!, 209
 - operator!=, 209
 - operator<, 214
 - operator<<, 219
 - operator<=, 214
 - operator>, 218
 - operator>=, 219
 - operator*, 210, 211
 - operator+, 211
 - operator-, 212
 - operator->, 213
 - operator/, 213
 - operator=, 215
 - operator==, 215–218
 - operator%, 210
- Tang::HtmlEscape, 220
 - get_next_token, 222
 - HtmlEscape, 221
- Tang::location, 222
- Tang::position, 224
- Tang::Program, 225
 - addBreak, 228
 - addBytecode, 228
 - addContinue, 228
 - addIdentifier, 228
 - addIdentifierAssigned, 229
 - addString, 229
 - CodeType, 227
 - dumpBytecode, 229
 - execute, 230
 - functionsDeclared, 234
 - getAst, 230
 - getBytecode, 230
 - getCode, 230
 - getIdentifiers, 231
 - getIdentifiersAssigned, 231
 - getResult, 231
 - getStrings, 231
 - popBreakStack, 232
 - popContinueStack, 232
 - Program, 227

- pushEnvironment, [233](#)
- Script, [227](#)
- setFunctionStackDeclaration, [233](#)
- setJumpTarget, [234](#)
- Template, [227](#)
- Tang::SingletonObjectPool< T >, [234](#)
 - get, [235](#)
 - getInstance, [235](#)
 - recycle, [235](#)
- Tang::TangBase, [236](#)
 - compileScript, [236](#)
 - TangBase, [236](#)
- Tang::TangScanner, [237](#)
 - get_next_token, [238](#)
 - TangScanner, [238](#)
- Tang::Unescape, [239](#)
 - get_next_token, [240](#)
 - Unescape, [240](#)
- Tang::UnicodeString, [241](#)
 - bytesLength, [242](#)
 - length, [242](#)
 - operator std::string, [242](#)
 - operator<, [243](#)
 - operator+, [242](#)
 - operator==, [243](#)
 - substr, [243](#)
 - UnicodeString, [241](#)
- TangBase
 - Tang::TangBase, [236](#)
- TangScanner
 - Tang::TangScanner, [238](#)
- Template
 - Tang::Program, [227](#)
- test/test.cpp, [322](#)
- test/testGarbageCollected.cpp, [324](#)
- test/testSingletonObjectPool.cpp, [325](#)
- test/testUnicodeString.cpp, [326](#)
- Type
 - Tang::AstNodeCast, [40](#)
 - Tang::AstNodePrint, [80](#)
- Unescape
 - Tang::Unescape, [240](#)
- unescape
 - unicodeString.hpp, [292](#)
- UnicodeString
 - Tang::UnicodeString, [241](#)
- unicodeString.hpp
 - htmlEscape, [292](#)
 - unescape, [292](#)