

Tang

0.1

Generated by Doxygen 1.9.1

1 Tang: A Template Language	1
1.1 Quick Description	1
1.2 Features	1
1.3 License	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	9
4.1 File List	9
5 Class Documentation	13
5.1 Tang::AstNode Class Reference	13
5.1.1 Detailed Description	16
5.1.2 Member Enumeration Documentation	16
5.1.2.1 PreprocessState	16
5.1.3 Constructor & Destructor Documentation	16
5.1.3.1 AstNode()	16
5.1.4 Member Function Documentation	17
5.1.4.1 compile()	17
5.1.4.2 compilePreprocess()	17
5.1.4.3 dump()	18
5.2 Tang::AstNodeArray Class Reference	18
5.2.1 Detailed Description	21
5.2.2 Member Enumeration Documentation	21
5.2.2.1 PreprocessState	21
5.2.3 Constructor & Destructor Documentation	21
5.2.3.1 AstNodeArray()	21
5.2.4 Member Function Documentation	22
5.2.4.1 compile()	22
5.2.4.2 compilePreprocess()	22
5.2.4.3 dump()	23
5.3 Tang::AstNodeAssign Class Reference	23
5.3.1 Detailed Description	25
5.3.2 Member Enumeration Documentation	25
5.3.2.1 PreprocessState	25
5.3.3 Constructor & Destructor Documentation	25
5.3.3.1 AstNodeAssign()	25
5.3.4 Member Function Documentation	26
5.3.4.1 compile()	26
5.3.4.2 compilePreprocess()	26

5.3.4.3 dump()	27
5.4 Tang::AstNodeBinary Class Reference	27
5.4.1 Detailed Description	29
5.4.2 Member Enumeration Documentation	29
5.4.2.1 Operation	29
5.4.2.2 PreprocessState	30
5.4.3 Constructor & Destructor Documentation	30
5.4.3.1 AstNodeBinary()	30
5.4.4 Member Function Documentation	30
5.4.4.1 compile()	30
5.4.4.2 compilePreprocess()	31
5.4.4.3 dump()	31
5.5 Tang::AstNodeBlock Class Reference	32
5.5.1 Detailed Description	34
5.5.2 Member Enumeration Documentation	34
5.5.2.1 PreprocessState	34
5.5.3 Constructor & Destructor Documentation	34
5.5.3.1 AstNodeBlock()	34
5.5.4 Member Function Documentation	35
5.5.4.1 compile()	35
5.5.4.2 compilePreprocess()	35
5.5.4.3 dump()	36
5.6 Tang::AstNodeBoolean Class Reference	36
5.6.1 Detailed Description	38
5.6.2 Member Enumeration Documentation	38
5.6.2.1 PreprocessState	38
5.6.3 Constructor & Destructor Documentation	38
5.6.3.1 AstNodeBoolean()	38
5.6.4 Member Function Documentation	39
5.6.4.1 compile()	39
5.6.4.2 compilePreprocess()	39
5.6.4.3 dump()	40
5.7 Tang::AstNodeBreak Class Reference	40
5.7.1 Detailed Description	42
5.7.2 Member Enumeration Documentation	42
5.7.2.1 PreprocessState	42
5.7.3 Constructor & Destructor Documentation	42
5.7.3.1 AstNodeBreak()	42
5.7.4 Member Function Documentation	42
5.7.4.1 compile()	43
5.7.4.2 compilePreprocess()	43
5.7.4.3 dump()	44

5.8 Tang::AstNodeCast Class Reference	44
5.8.1 Detailed Description	46
5.8.2 Member Enumeration Documentation	46
5.8.2.1 PreprocessState	46
5.8.2.2 Type	46
5.8.3 Constructor & Destructor Documentation	47
5.8.3.1 AstNodeCast()	47
5.8.4 Member Function Documentation	47
5.8.4.1 compile()	47
5.8.4.2 compilePreprocess()	48
5.8.4.3 dump()	48
5.9 Tang::AstNodeContinue Class Reference	48
5.9.1 Detailed Description	50
5.9.2 Member Enumeration Documentation	50
5.9.2.1 PreprocessState	50
5.9.3 Constructor & Destructor Documentation	50
5.9.3.1 AstNodeContinue()	51
5.9.4 Member Function Documentation	51
5.9.4.1 compile()	51
5.9.4.2 compilePreprocess()	52
5.9.4.3 dump()	52
5.10 Tang::AstNodeDoWhile Class Reference	52
5.10.1 Detailed Description	55
5.10.2 Member Enumeration Documentation	55
5.10.2.1 PreprocessState	55
5.10.3 Constructor & Destructor Documentation	55
5.10.3.1 AstNodeDoWhile()	55
5.10.4 Member Function Documentation	56
5.10.4.1 compile()	56
5.10.4.2 compilePreprocess()	56
5.10.4.3 dump()	57
5.11 Tang::AstNodeFloat Class Reference	57
5.11.1 Detailed Description	59
5.11.2 Member Enumeration Documentation	59
5.11.2.1 PreprocessState	59
5.11.3 Constructor & Destructor Documentation	59
5.11.3.1 AstNodeFloat()	59
5.11.4 Member Function Documentation	60
5.11.4.1 compile()	60
5.11.4.2 compilePreprocess()	60
5.11.4.3 dump()	61
5.12 Tang::AstNodeFor Class Reference	61

5.12.1 Detailed Description	63
5.12.2 Member Enumeration Documentation	63
5.12.2.1 PreprocessState	63
5.12.3 Constructor & Destructor Documentation	63
5.12.3.1 AstNodeFor()	64
5.12.4 Member Function Documentation	64
5.12.4.1 compile()	64
5.12.4.2 compilePreprocess()	65
5.12.4.3 dump()	65
5.13 Tang::AstNodeFunctionCall Class Reference	66
5.13.1 Detailed Description	67
5.13.2 Member Enumeration Documentation	67
5.13.2.1 PreprocessState	67
5.13.3 Constructor & Destructor Documentation	67
5.13.3.1 AstNodeFunctionCall()	68
5.13.4 Member Function Documentation	68
5.13.4.1 compile()	68
5.13.4.2 compilePreprocess()	68
5.13.4.3 dump()	69
5.14 Tang::AstNodeFunctionDeclaration Class Reference	69
5.14.1 Detailed Description	70
5.14.2 Member Enumeration Documentation	70
5.14.2.1 PreprocessState	70
5.14.3 Constructor & Destructor Documentation	71
5.14.3.1 AstNodeFunctionDeclaration()	71
5.14.4 Member Function Documentation	71
5.14.4.1 compile()	71
5.14.4.2 compilePreprocess()	72
5.14.4.3 dump()	73
5.15 Tang::AstNodeIdentifier Class Reference	73
5.15.1 Detailed Description	75
5.15.2 Member Enumeration Documentation	75
5.15.2.1 PreprocessState	75
5.15.3 Constructor & Destructor Documentation	75
5.15.3.1 AstNodeIdentifier()	75
5.15.4 Member Function Documentation	76
5.15.4.1 compile()	76
5.15.4.2 compilePreprocess()	76
5.15.4.3 dump()	77
5.16 Tang::AstNodeIfElse Class Reference	77
5.16.1 Detailed Description	80
5.16.2 Member Enumeration Documentation	80

5.16.2.1 PreprocessState	80
5.16.3 Constructor & Destructor Documentation	80
5.16.3.1 AstNodeIfElse() [1/2]	81
5.16.3.2 AstNodeIfElse() [2/2]	81
5.16.4 Member Function Documentation	81
5.16.4.1 compile()	81
5.16.4.2 compilePreprocess()	82
5.16.4.3 dump()	82
5.17 Tang::AstNodeIndex Class Reference	83
5.17.1 Detailed Description	85
5.17.2 Member Enumeration Documentation	85
5.17.2.1 PreprocessState	85
5.17.3 Constructor & Destructor Documentation	85
5.17.3.1 AstNodeIndex()	85
5.17.4 Member Function Documentation	86
5.17.4.1 compile()	86
5.17.4.2 compilePreprocess()	86
5.17.4.3 dump()	87
5.17.4.4 getCollection()	87
5.17.4.5 getIndex()	87
5.18 Tang::AstNodeInteger Class Reference	88
5.18.1 Detailed Description	90
5.18.2 Member Enumeration Documentation	90
5.18.2.1 PreprocessState	90
5.18.3 Constructor & Destructor Documentation	90
5.18.3.1 AstNodeInteger()	90
5.18.4 Member Function Documentation	91
5.18.4.1 compile()	91
5.18.4.2 compilePreprocess()	91
5.18.4.3 dump()	92
5.19 Tang::AstNodePrint Class Reference	92
5.19.1 Detailed Description	94
5.19.2 Member Enumeration Documentation	94
5.19.2.1 PreprocessState	94
5.19.2.2 Type	94
5.19.3 Constructor & Destructor Documentation	95
5.19.3.1 AstNodePrint()	95
5.19.4 Member Function Documentation	95
5.19.4.1 compile()	95
5.19.4.2 compilePreprocess()	96
5.19.4.3 dump()	96
5.20 Tang::AstNodeRangedFor Class Reference	96

5.20.1 Detailed Description	98
5.20.2 Member Enumeration Documentation	98
5.20.2.1 PreprocessState	98
5.20.3 Constructor & Destructor Documentation	98
5.20.3.1 AstNodeRangedFor()	98
5.20.4 Member Function Documentation	98
5.20.4.1 compile()	99
5.20.4.2 compilePreprocess()	99
5.20.4.3 dump()	100
5.21 Tang::AstNodeReturn Class Reference	100
5.21.1 Detailed Description	102
5.21.2 Member Enumeration Documentation	102
5.21.2.1 PreprocessState	102
5.21.3 Constructor & Destructor Documentation	102
5.21.3.1 AstNodeReturn()	103
5.21.4 Member Function Documentation	103
5.21.4.1 compile()	103
5.21.4.2 compilePreprocess()	104
5.21.4.3 dump()	104
5.22 Tang::AstNodeSlice Class Reference	104
5.22.1 Detailed Description	107
5.22.2 Member Enumeration Documentation	107
5.22.2.1 PreprocessState	107
5.22.3 Constructor & Destructor Documentation	107
5.22.3.1 AstNodeSlice()	108
5.22.4 Member Function Documentation	108
5.22.4.1 compile()	108
5.22.4.2 compilePreprocess()	109
5.22.4.3 dump()	109
5.23 Tang::AstNodeString Class Reference	109
5.23.1 Detailed Description	111
5.23.2 Member Enumeration Documentation	111
5.23.2.1 PreprocessState	111
5.23.3 Constructor & Destructor Documentation	112
5.23.3.1 AstNodeString()	112
5.23.4 Member Function Documentation	112
5.23.4.1 compile()	112
5.23.4.2 compileLiteral()	113
5.23.4.3 compilePreprocess()	113
5.23.4.4 dump()	114
5.24 Tang::AstNodeTernary Class Reference	114
5.24.1 Detailed Description	117

5.24.2 Member Enumeration Documentation	117
5.24.2.1 PreprocessState	117
5.24.3 Constructor & Destructor Documentation	117
5.24.3.1 AstNodeTernary()	117
5.24.4 Member Function Documentation	118
5.24.4.1 compile()	118
5.24.4.2 compilePreprocess()	119
5.24.4.3 dump()	119
5.25 Tang::AstNodeUnary Class Reference	119
5.25.1 Detailed Description	121
5.25.2 Member Enumeration Documentation	121
5.25.2.1 Operator	121
5.25.2.2 PreprocessState	122
5.25.3 Constructor & Destructor Documentation	122
5.25.3.1 AstNodeUnary()	122
5.25.4 Member Function Documentation	122
5.25.4.1 compile()	122
5.25.4.2 compilePreprocess()	123
5.25.4.3 dump()	123
5.26 Tang::AstNodeWhile Class Reference	124
5.26.1 Detailed Description	126
5.26.2 Member Enumeration Documentation	126
5.26.2.1 PreprocessState	126
5.26.3 Constructor & Destructor Documentation	126
5.26.3.1 AstNodeWhile()	126
5.26.4 Member Function Documentation	127
5.26.4.1 compile()	127
5.26.4.2 compilePreprocess()	127
5.26.4.3 dump()	128
5.27 Tang::ComputedExpression Class Reference	128
5.27.1 Detailed Description	130
5.27.2 Member Function Documentation	131
5.27.2.1 __add()	131
5.27.2.2 __asCode()	131
5.27.2.3 __assign_index()	131
5.27.2.4 __boolean()	132
5.27.2.5 __divide()	132
5.27.2.6 __equal()	132
5.27.2.7 __float()	133
5.27.2.8 __getIterator()	133
5.27.2.9 __index()	133
5.27.2.10 __integer()	134

5.27.2.11	__iteratorNext()	134
5.27.2.12	__lessThan()	134
5.27.2.13	__modulo()	135
5.27.2.14	__multiply()	135
5.27.2.15	__negative()	136
5.27.2.16	__not()	136
5.27.2.17	__slice()	136
5.27.2.18	__string()	137
5.27.2.19	__subtract()	137
5.27.2.20	dump()	137
5.27.2.21	is_equal() [1/6]	138
5.27.2.22	is_equal() [2/6]	138
5.27.2.23	is_equal() [3/6]	139
5.27.2.24	is_equal() [4/6]	139
5.27.2.25	is_equal() [5/6]	139
5.27.2.26	is_equal() [6/6]	140
5.27.2.27	isCopyNeeded()	140
5.27.2.28	makeCopy()	141
5.28	Tang::ComputedExpressionArray Class Reference	141
5.28.1	Detailed Description	143
5.28.2	Constructor & Destructor Documentation	144
5.28.2.1	ComputedExpressionArray()	144
5.28.3	Member Function Documentation	144
5.28.3.1	__add()	144
5.28.3.2	__asCode()	144
5.28.3.3	__assign_index()	145
5.28.3.4	__boolean()	145
5.28.3.5	__divide()	145
5.28.3.6	__equal()	146
5.28.3.7	__float()	146
5.28.3.8	__getIterator()	146
5.28.3.9	__index()	147
5.28.3.10	__integer()	147
5.28.3.11	__iteratorNext()	148
5.28.3.12	__lessThan()	148
5.28.3.13	__modulo()	148
5.28.3.14	__multiply()	149
5.28.3.15	__negative()	149
5.28.3.16	__not()	149
5.28.3.17	__slice()	150
5.28.3.18	__string()	150
5.28.3.19	__subtract()	151

5.28.3.20 dump()	151
5.28.3.21 is_equal() [1/6]	152
5.28.3.22 is_equal() [2/6]	152
5.28.3.23 is_equal() [3/6]	152
5.28.3.24 is_equal() [4/6]	153
5.28.3.25 is_equal() [5/6]	153
5.28.3.26 is_equal() [6/6]	153
5.28.3.27 isCopyNeeded()	154
5.28.3.28 makeCopy()	154
5.29 Tang::ComputedExpressionBoolean Class Reference	155
5.29.1 Detailed Description	157
5.29.2 Constructor & Destructor Documentation	157
5.29.2.1 ComputedExpressionBoolean()	157
5.29.3 Member Function Documentation	157
5.29.3.1 __add()	157
5.29.3.2 __asCode()	158
5.29.3.3 __assign_index()	158
5.29.3.4 __boolean()	158
5.29.3.5 __divide()	158
5.29.3.6 __equal()	159
5.29.3.7 __float()	159
5.29.3.8 __getIterator()	160
5.29.3.9 __index()	160
5.29.3.10 __integer()	160
5.29.3.11 __iteratorNext()	160
5.29.3.12 __lessThan()	161
5.29.3.13 __modulo()	161
5.29.3.14 __multiply()	162
5.29.3.15 __negative()	162
5.29.3.16 __not()	162
5.29.3.17 __slice()	162
5.29.3.18 __string()	163
5.29.3.19 __subtract()	163
5.29.3.20 dump()	164
5.29.3.21 is_equal() [1/6]	164
5.29.3.22 is_equal() [2/6]	164
5.29.3.23 is_equal() [3/6]	165
5.29.3.24 is_equal() [4/6]	165
5.29.3.25 is_equal() [5/6]	165
5.29.3.26 is_equal() [6/6]	166
5.29.3.27 isCopyNeeded()	166
5.29.3.28 makeCopy()	166

5.30 Tang::ComputedExpressionCompiledFunction Class Reference	167
5.30.1 Detailed Description	169
5.30.2 Constructor & Destructor Documentation	169
5.30.2.1 ComputedExpressionCompiledFunction()	169
5.30.3 Member Function Documentation	169
5.30.3.1 __add()	169
5.30.3.2 __asCode()	170
5.30.3.3 __assign_index()	170
5.30.3.4 __boolean()	170
5.30.3.5 __divide()	171
5.30.3.6 __equal()	171
5.30.3.7 __float()	172
5.30.3.8 __getIterator()	172
5.30.3.9 __index()	172
5.30.3.10 __integer()	173
5.30.3.11 __iteratorNext()	173
5.30.3.12 __lessThan()	173
5.30.3.13 __modulo()	174
5.30.3.14 __multiply()	174
5.30.3.15 __negative()	175
5.30.3.16 __not()	175
5.30.3.17 __slice()	175
5.30.3.18 __string()	176
5.30.3.19 __subtract()	176
5.30.3.20 dump()	176
5.30.3.21 is_equal() [1/6]	177
5.30.3.22 is_equal() [2/6]	177
5.30.3.23 is_equal() [3/6]	177
5.30.3.24 is_equal() [4/6]	178
5.30.3.25 is_equal() [5/6]	178
5.30.3.26 is_equal() [6/6]	178
5.30.3.27 isCopyNeeded()	179
5.30.3.28 makeCopy()	179
5.31 Tang::ComputedExpressionError Class Reference	180
5.31.1 Detailed Description	183
5.31.2 Constructor & Destructor Documentation	183
5.31.2.1 ComputedExpressionError()	183
5.31.3 Member Function Documentation	183
5.31.3.1 __add()	183
5.31.3.2 __asCode()	183
5.31.3.3 __assign_index()	184
5.31.3.4 __boolean()	184

5.31.3.5 <code>__divide()</code>	184
5.31.3.6 <code>__equal()</code>	185
5.31.3.7 <code>__float()</code>	185
5.31.3.8 <code>__getIterator()</code>	185
5.31.3.9 <code>__index()</code>	186
5.31.3.10 <code>__integer()</code>	186
5.31.3.11 <code>__iteratorNext()</code>	186
5.31.3.12 <code>__lessThan()</code>	187
5.31.3.13 <code>__modulo()</code>	187
5.31.3.14 <code>__multiply()</code>	188
5.31.3.15 <code>__negative()</code>	188
5.31.3.16 <code>__not()</code>	188
5.31.3.17 <code>__slice()</code>	188
5.31.3.18 <code>__string()</code>	189
5.31.3.19 <code>__subtract()</code>	189
5.31.3.20 <code>dump()</code>	190
5.31.3.21 <code>is_equal()</code> [1/6]	190
5.31.3.22 <code>is_equal()</code> [2/6]	190
5.31.3.23 <code>is_equal()</code> [3/6]	191
5.31.3.24 <code>is_equal()</code> [4/6]	191
5.31.3.25 <code>is_equal()</code> [5/6]	191
5.31.3.26 <code>is_equal()</code> [6/6]	192
5.31.3.27 <code>isCopyNeeded()</code>	192
5.31.3.28 <code>makeCopy()</code>	193
5.32 Tang::ComputedExpressionFloat Class Reference	193
5.32.1 Detailed Description	195
5.32.2 Constructor & Destructor Documentation	195
5.32.2.1 <code>ComputedExpressionFloat()</code>	195
5.32.3 Member Function Documentation	195
5.32.3.1 <code>__add()</code>	195
5.32.3.2 <code>__asCode()</code>	197
5.32.3.3 <code>__assign_index()</code>	197
5.32.3.4 <code>__boolean()</code>	198
5.32.3.5 <code>__divide()</code>	198
5.32.3.6 <code>__equal()</code>	199
5.32.3.7 <code>__float()</code>	199
5.32.3.8 <code>__getIterator()</code>	199
5.32.3.9 <code>__index()</code>	200
5.32.3.10 <code>__integer()</code>	200
5.32.3.11 <code>__iteratorNext()</code>	200
5.32.3.12 <code>__lessThan()</code>	201
5.32.3.13 <code>__modulo()</code>	201

5.32.3.14	<code>__multiply()</code>	202
5.32.3.15	<code>__negative()</code>	202
5.32.3.16	<code>__not()</code>	203
5.32.3.17	<code>__slice()</code>	203
5.32.3.18	<code>__string()</code>	203
5.32.3.19	<code>__subtract()</code>	204
5.32.3.20	<code>dump()</code>	205
5.32.3.21	<code>getValue()</code>	205
5.32.3.22	<code>is_equal()</code> [1/6]	205
5.32.3.23	<code>is_equal()</code> [2/6]	205
5.32.3.24	<code>is_equal()</code> [3/6]	206
5.32.3.25	<code>is_equal()</code> [4/6]	206
5.32.3.26	<code>is_equal()</code> [5/6]	207
5.32.3.27	<code>is_equal()</code> [6/6]	207
5.32.3.28	<code>isCopyNeeded()</code>	207
5.32.3.29	<code>makeCopy()</code>	208
5.33	Tang::ComputedExpressionInteger Class Reference	208
5.33.1	Detailed Description	210
5.33.2	Constructor & Destructor Documentation	210
5.33.2.1	<code>ComputedExpressionInteger()</code>	210
5.33.3	Member Function Documentation	210
5.33.3.1	<code>__add()</code>	210
5.33.3.2	<code>__asCode()</code>	212
5.33.3.3	<code>__assign_index()</code>	212
5.33.3.4	<code>__boolean()</code>	213
5.33.3.5	<code>__divide()</code>	213
5.33.3.6	<code>__equal()</code>	214
5.33.3.7	<code>__float()</code>	214
5.33.3.8	<code>__getIterator()</code>	214
5.33.3.9	<code>__index()</code>	215
5.33.3.10	<code>__integer()</code>	215
5.33.3.11	<code>__iteratorNext()</code>	215
5.33.3.12	<code>__lessThan()</code>	216
5.33.3.13	<code>__modulo()</code>	216
5.33.3.14	<code>__multiply()</code>	217
5.33.3.15	<code>__negative()</code>	218
5.33.3.16	<code>__not()</code>	218
5.33.3.17	<code>__slice()</code>	218
5.33.3.18	<code>__string()</code>	219
5.33.3.19	<code>__subtract()</code>	219
5.33.3.20	<code>dump()</code>	220
5.33.3.21	<code>getValue()</code>	220

5.33.3.22 is_equal() [1/6]	220
5.33.3.23 is_equal() [2/6]	221
5.33.3.24 is_equal() [3/6]	221
5.33.3.25 is_equal() [4/6]	221
5.33.3.26 is_equal() [5/6]	222
5.33.3.27 is_equal() [6/6]	222
5.33.3.28 isCopyNeeded()	223
5.33.3.29 makeCopy()	223
5.34 Tang::ComputedExpressionIterator Class Reference	223
5.34.1 Detailed Description	225
5.34.2 Constructor & Destructor Documentation	225
5.34.2.1 ComputedExpressionIterator()	226
5.34.3 Member Function Documentation	226
5.34.3.1 __add()	226
5.34.3.2 __asCode()	226
5.34.3.3 __assign_index()	227
5.34.3.4 __boolean()	227
5.34.3.5 __divide()	227
5.34.3.6 __equal()	228
5.34.3.7 __float()	228
5.34.3.8 __getIterator()	228
5.34.3.9 __index()	229
5.34.3.10 __integer()	229
5.34.3.11 __iteratorNext()	229
5.34.3.12 __lessThan()	230
5.34.3.13 __modulo()	230
5.34.3.14 __multiply()	231
5.34.3.15 __negative()	231
5.34.3.16 __not()	231
5.34.3.17 __slice()	232
5.34.3.18 __string()	232
5.34.3.19 __subtract()	232
5.34.3.20 dump()	233
5.34.3.21 is_equal() [1/6]	233
5.34.3.22 is_equal() [2/6]	234
5.34.3.23 is_equal() [3/6]	235
5.34.3.24 is_equal() [4/6]	235
5.34.3.25 is_equal() [5/6]	236
5.34.3.26 is_equal() [6/6]	236
5.34.3.27 isCopyNeeded()	236
5.34.3.28 makeCopy()	237
5.35 Tang::ComputedExpressionIteratorEnd Class Reference	237

5.35.1 Detailed Description	239
5.35.2 Member Function Documentation	239
5.35.2.1 __add()	239
5.35.2.2 __asCode()	239
5.35.2.3 __assign_index()	240
5.35.2.4 __boolean()	240
5.35.2.5 __divide()	240
5.35.2.6 __equal()	241
5.35.2.7 __float()	241
5.35.2.8 __getIterator()	241
5.35.2.9 __index()	242
5.35.2.10 __integer()	242
5.35.2.11 __iteratorNext()	242
5.35.2.12 __lessThan()	243
5.35.2.13 __modulo()	243
5.35.2.14 __multiply()	244
5.35.2.15 __negative()	244
5.35.2.16 __not()	244
5.35.2.17 __slice()	244
5.35.2.18 __string()	245
5.35.2.19 __subtract()	245
5.35.2.20 dump()	246
5.35.2.21 is_equal() [1/6]	246
5.35.2.22 is_equal() [2/6]	246
5.35.2.23 is_equal() [3/6]	247
5.35.2.24 is_equal() [4/6]	247
5.35.2.25 is_equal() [5/6]	247
5.35.2.26 is_equal() [6/6]	248
5.35.2.27 isCopyNeeded()	248
5.35.2.28 makeCopy()	249
5.36 Tang::ComputedExpressionString Class Reference	249
5.36.1 Detailed Description	251
5.36.2 Constructor & Destructor Documentation	251
5.36.2.1 ComputedExpressionString()	251
5.36.3 Member Function Documentation	252
5.36.3.1 __add()	252
5.36.3.2 __asCode()	252
5.36.3.3 __assign_index()	253
5.36.3.4 __boolean()	253
5.36.3.5 __divide()	254
5.36.3.6 __equal()	254
5.36.3.7 __float()	255

5.36.3.8	__getIterator()	255
5.36.3.9	__index()	256
5.36.3.10	__integer()	256
5.36.3.11	__iteratorNext()	256
5.36.3.12	__lessThan()	257
5.36.3.13	__modulo()	258
5.36.3.14	__multiply()	258
5.36.3.15	__negative()	258
5.36.3.16	__not()	259
5.36.3.17	__slice()	259
5.36.3.18	__string()	260
5.36.3.19	__subtract()	260
5.36.3.20	dump()	261
5.36.3.21	is_equal() [1/6]	261
5.36.3.22	is_equal() [2/6]	261
5.36.3.23	is_equal() [3/6]	262
5.36.3.24	is_equal() [4/6]	262
5.36.3.25	is_equal() [5/6]	263
5.36.3.26	is_equal() [6/6]	263
5.36.3.27	isCopyNeeded()	263
5.36.3.28	makeCopy()	264
5.37	Tang::Error Class Reference	264
5.37.1	Detailed Description	265
5.37.2	Constructor & Destructor Documentation	265
5.37.2.1	Error() [1/2]	265
5.37.2.2	Error() [2/2]	265
5.37.3	Friends And Related Function Documentation	266
5.37.3.1	operator<<	266
5.38	Tang::GarbageCollected Class Reference	266
5.38.1	Detailed Description	269
5.38.2	Constructor & Destructor Documentation	269
5.38.2.1	GarbageCollected() [1/3]	269
5.38.2.2	GarbageCollected() [2/3]	269
5.38.2.3	~GarbageCollected()	269
5.38.2.4	GarbageCollected() [3/3]	270
5.38.3	Member Function Documentation	270
5.38.3.1	isCopyNeeded()	270
5.38.3.2	make()	270
5.38.3.3	makeCopy()	271
5.38.3.4	operator"!(())	271
5.38.3.5	operator"!=(())	272
5.38.3.6	operator%()	272

5.38.3.7 operator*() [1/2]	273
5.38.3.8 operator*() [2/2]	273
5.38.3.9 operator+()	274
5.38.3.10 operator-() [1/2]	274
5.38.3.11 operator-() [2/2]	275
5.38.3.12 operator->()	275
5.38.3.13 operator/()	276
5.38.3.14 operator<()	276
5.38.3.15 operator<=()	277
5.38.3.16 operator=() [1/2]	277
5.38.3.17 operator=() [2/2]	277
5.38.3.18 operator==(1/8)	279
5.38.3.19 operator==(2/8)	279
5.38.3.20 operator==(3/8)	279
5.38.3.21 operator==(4/8)	280
5.38.3.22 operator==(5/8)	280
5.38.3.23 operator==(6/8)	281
5.38.3.24 operator==(7/8)	281
5.38.3.25 operator==(8/8)	281
5.38.3.26 operator>()	282
5.38.3.27 operator>=()	282
5.38.4 Friends And Related Function Documentation	283
5.38.4.1 operator<<	283
5.39 Tang::HtmlEscape Class Reference	283
5.39.1 Detailed Description	284
5.39.2 Constructor & Destructor Documentation	284
5.39.2.1 HtmlEscape()	285
5.39.3 Member Function Documentation	285
5.39.3.1 get_next_token()	285
5.40 Tang::HtmlEscapeAscii Class Reference	285
5.40.1 Detailed Description	286
5.40.2 Constructor & Destructor Documentation	286
5.40.2.1 HtmlEscapeAscii()	287
5.40.3 Member Function Documentation	287
5.40.3.1 get_next_token()	287
5.41 Tang::location Class Reference	287
5.41.1 Detailed Description	289
5.42 Tang::position Class Reference	289
5.42.1 Detailed Description	290
5.43 Tang::Program Class Reference	290
5.43.1 Detailed Description	293
5.43.2 Member Enumeration Documentation	293

5.43.2.1 CodeType	293
5.43.3 Constructor & Destructor Documentation	293
5.43.3.1 Program()	293
5.43.4 Member Function Documentation	294
5.43.4.1 addBreak()	294
5.43.4.2 addBytecode()	294
5.43.4.3 addContinue()	295
5.43.4.4 addIdentifier()	295
5.43.4.5 addIdentifierAssigned()	295
5.43.4.6 addString()	295
5.43.4.7 dumpBytecode()	296
5.43.4.8 execute()	296
5.43.4.9 getAst()	296
5.43.4.10 getBytecode()	297
5.43.4.11 getCode()	297
5.43.4.12 getIdentifiers()	297
5.43.4.13 getIdentifiersAssigned()	297
5.43.4.14 getResult()	298
5.43.4.15 getStrings()	298
5.43.4.16 popBreakStack()	298
5.43.4.17 popContinueStack()	299
5.43.4.18 pushEnvironment()	299
5.43.4.19 setFunctionStackDeclaration()	300
5.43.4.20 setJumpTarget()	300
5.43.5 Member Data Documentation	300
5.43.5.1 functionsDeclared	300
5.44 Tang::SingletonObjectPool< T > Class Template Reference	301
5.44.1 Detailed Description	302
5.44.2 Member Function Documentation	302
5.44.2.1 get()	302
5.44.2.2 getInstance()	303
5.44.2.3 recycle()	303
5.44.3 Member Data Documentation	303
5.44.3.1 currentIndex	303
5.44.3.2 currentRecycledIndex	303
5.45 Tang::TangBase Class Reference	304
5.45.1 Detailed Description	304
5.45.2 Constructor & Destructor Documentation	304
5.45.2.1 TangBase()	304
5.45.3 Member Function Documentation	304
5.45.3.1 compileScript()	304
5.46 Tang::TangScanner Class Reference	305

5.46.1 Detailed Description	306
5.46.2 Constructor & Destructor Documentation	307
5.46.2.1 TangScanner()	307
5.46.3 Member Function Documentation	307
5.46.3.1 get_next_token()	307
5.47 Tang::Unescape Class Reference	308
5.47.1 Detailed Description	308
5.47.2 Constructor & Destructor Documentation	309
5.47.2.1 Unescape()	309
5.47.3 Member Function Documentation	309
5.47.3.1 get_next_token()	309
5.48 Tang::UnicodeString Class Reference	310
5.48.1 Detailed Description	311
5.48.2 Constructor & Destructor Documentation	311
5.48.2.1 UnicodeString()	311
5.48.3 Member Function Documentation	311
5.48.3.1 bytesLength()	311
5.48.3.2 length()	312
5.48.3.3 operator std::string()	312
5.48.3.4 operator+()	312
5.48.3.5 operator<()	313
5.48.3.6 operator==(())	313
5.48.3.7 substr()	313
6 File Documentation	315
6.1 build/generated/location.hh File Reference	315
6.1.1 Detailed Description	316
6.1.2 Function Documentation	316
6.1.2.1 operator<<() [1/2]	316
6.1.2.2 operator<<() [2/2]	317
6.2 include/astNode.hpp File Reference	317
6.2.1 Detailed Description	318
6.3 include/astNodeArray.hpp File Reference	318
6.3.1 Detailed Description	319
6.4 include/astNodeAssign.hpp File Reference	319
6.4.1 Detailed Description	320
6.5 include/astNodeBinary.hpp File Reference	320
6.5.1 Detailed Description	321
6.6 include/astNodeBlock.hpp File Reference	321
6.6.1 Detailed Description	322
6.7 include/astNodeBoolean.hpp File Reference	322
6.7.1 Detailed Description	323

6.8 include/astNodeBreak.hpp File Reference	323
6.8.1 Detailed Description	324
6.9 include/astNodeCast.hpp File Reference	324
6.9.1 Detailed Description	325
6.10 include/astNodeContinue.hpp File Reference	325
6.10.1 Detailed Description	326
6.11 include/astNodeDoWhile.hpp File Reference	326
6.11.1 Detailed Description	327
6.12 include/astNodeFloat.hpp File Reference	327
6.12.1 Detailed Description	328
6.13 include/astNodeFor.hpp File Reference	328
6.13.1 Detailed Description	329
6.14 include/astNodeFunctionCall.hpp File Reference	329
6.14.1 Detailed Description	330
6.15 include/astNodeFunctionDeclaration.hpp File Reference	330
6.15.1 Detailed Description	331
6.16 include/astNodeIdentifier.hpp File Reference	331
6.16.1 Detailed Description	332
6.17 include/astNodeIfElse.hpp File Reference	332
6.17.1 Detailed Description	333
6.18 include/astNodeIndex.hpp File Reference	333
6.18.1 Detailed Description	334
6.19 include/astNodeInteger.hpp File Reference	334
6.19.1 Detailed Description	335
6.20 include/astNodePrint.hpp File Reference	335
6.20.1 Detailed Description	336
6.21 include/astNodeRangedFor.hpp File Reference	336
6.21.1 Detailed Description	337
6.22 include/astNodeReturn.hpp File Reference	337
6.22.1 Detailed Description	338
6.23 include/astNodeSlice.hpp File Reference	338
6.23.1 Detailed Description	339
6.24 include/astNodeString.hpp File Reference	339
6.24.1 Detailed Description	340
6.25 include/astNodeTernary.hpp File Reference	340
6.25.1 Detailed Description	341
6.26 include/astNodeUnary.hpp File Reference	341
6.26.1 Detailed Description	342
6.27 include/astNodeWhile.hpp File Reference	342
6.27.1 Detailed Description	343
6.28 include/computedExpression.hpp File Reference	343
6.28.1 Detailed Description	343

6.29 include/computedExpressionArray.hpp File Reference	344
6.29.1 Detailed Description	344
6.30 include/computedExpressionBoolean.hpp File Reference	345
6.30.1 Detailed Description	345
6.31 include/computedExpressionCompiledFunction.hpp File Reference	346
6.31.1 Detailed Description	346
6.32 include/computedExpressionError.hpp File Reference	347
6.32.1 Detailed Description	347
6.33 include/computedExpressionFloat.hpp File Reference	348
6.33.1 Detailed Description	348
6.34 include/computedExpressionInteger.hpp File Reference	349
6.34.1 Detailed Description	349
6.35 include/computedExpressionIterator.hpp File Reference	350
6.35.1 Detailed Description	350
6.36 include/computedExpressionIteratorEnd.hpp File Reference	351
6.36.1 Detailed Description	351
6.37 include/computedExpressionString.hpp File Reference	352
6.37.1 Detailed Description	352
6.38 include/error.hpp File Reference	352
6.38.1 Detailed Description	353
6.39 include/garbageCollected.hpp File Reference	353
6.39.1 Detailed Description	354
6.40 include/htmlEscape.hpp File Reference	354
6.40.1 Detailed Description	355
6.41 include/htmlEscapeAscii.hpp File Reference	356
6.41.1 Detailed Description	356
6.42 include/macros.hpp File Reference	357
6.42.1 Detailed Description	357
6.43 include/opcode.hpp File Reference	357
6.43.1 Detailed Description	358
6.43.2 Enumeration Type Documentation	358
6.43.2.1 Opcode	358
6.44 include/program.hpp File Reference	359
6.44.1 Detailed Description	360
6.45 include/singletonObjectPool.hpp File Reference	360
6.45.1 Detailed Description	361
6.46 include/tang.hpp File Reference	361
6.46.1 Detailed Description	361
6.47 include/tangBase.hpp File Reference	362
6.47.1 Detailed Description	362
6.48 include/tangScanner.hpp File Reference	363
6.48.1 Detailed Description	364

6.49 include/unescape.hpp File Reference	364
6.49.1 Detailed Description	365
6.50 include/unicodeString.hpp File Reference	365
6.50.1 Detailed Description	366
6.50.2 Function Documentation	366
6.50.2.1 htmlEscape()	366
6.50.2.2 htmlEscapeAscii()	366
6.50.2.3 unescape()	367
6.51 src/astNode.cpp File Reference	368
6.51.1 Detailed Description	368
6.52 src/astNodeArray.cpp File Reference	368
6.52.1 Detailed Description	369
6.53 src/astNodeAssign.cpp File Reference	369
6.53.1 Detailed Description	370
6.54 src/astNodeBinary.cpp File Reference	370
6.54.1 Detailed Description	371
6.55 src/astNodeBlock.cpp File Reference	371
6.55.1 Detailed Description	371
6.56 src/astNodeBoolean.cpp File Reference	371
6.56.1 Detailed Description	372
6.57 src/astNodeBreak.cpp File Reference	372
6.57.1 Detailed Description	373
6.58 src/astNodeCast.cpp File Reference	373
6.58.1 Detailed Description	373
6.59 src/astNodeContinue.cpp File Reference	373
6.59.1 Detailed Description	374
6.60 src/astNodeDoWhile.cpp File Reference	374
6.60.1 Detailed Description	375
6.61 src/astNodeFloat.cpp File Reference	375
6.61.1 Detailed Description	376
6.62 src/astNodeFor.cpp File Reference	376
6.62.1 Detailed Description	376
6.63 src/astNodeFunctionCall.cpp File Reference	376
6.63.1 Detailed Description	377
6.64 src/astNodeFunctionDeclaration.cpp File Reference	377
6.64.1 Detailed Description	378
6.65 src/astNodeIdentifier.cpp File Reference	378
6.65.1 Detailed Description	379
6.66 src/astNodeIfElse.cpp File Reference	379
6.66.1 Detailed Description	379
6.67 src/astNodeIndex.cpp File Reference	379
6.67.1 Detailed Description	380

6.68	src/astNodeInteger.cpp File Reference	380
6.68.1	Detailed Description	381
6.69	src/astNodePrint.cpp File Reference	381
6.69.1	Detailed Description	381
6.70	src/astNodeRangedFor.cpp File Reference	381
6.70.1	Detailed Description	382
6.71	src/astNodeReturn.cpp File Reference	382
6.71.1	Detailed Description	383
6.72	src/astNodeSlice.cpp File Reference	383
6.72.1	Detailed Description	384
6.73	src/astNodeString.cpp File Reference	384
6.73.1	Detailed Description	385
6.74	src/astNodeTernary.cpp File Reference	385
6.74.1	Detailed Description	386
6.75	src/astNodeUnary.cpp File Reference	386
6.75.1	Detailed Description	386
6.76	src/astNodeWhile.cpp File Reference	386
6.76.1	Detailed Description	387
6.77	src/computedExpression.cpp File Reference	387
6.77.1	Detailed Description	388
6.78	src/computedExpressionArray.cpp File Reference	388
6.78.1	Detailed Description	389
6.79	src/computedExpressionBoolean.cpp File Reference	389
6.79.1	Detailed Description	389
6.80	src/computedExpressionCompiledFunction.cpp File Reference	389
6.80.1	Detailed Description	390
6.81	src/computedExpressionError.cpp File Reference	390
6.81.1	Detailed Description	391
6.82	src/computedExpressionFloat.cpp File Reference	391
6.82.1	Detailed Description	391
6.83	src/computedExpressionInteger.cpp File Reference	391
6.83.1	Detailed Description	392
6.84	src/computedExpressionIterator.cpp File Reference	392
6.84.1	Detailed Description	392
6.85	src/computedExpressionIteratorEnd.cpp File Reference	393
6.85.1	Detailed Description	393
6.86	src/computedExpressionString.cpp File Reference	393
6.86.1	Detailed Description	394
6.87	src/error.cpp File Reference	394
6.87.1	Detailed Description	394
6.87.2	Function Documentation	394
6.87.2.1	operator<<()	394

6.88 src/program-dumpBytecode.cpp File Reference	395
6.88.1 Detailed Description	395
6.88.2 Macro Definition Documentation	395
6.88.2.1 DUMPPROGRAMCHECK	396
6.89 src/program-execute.cpp File Reference	396
6.89.1 Detailed Description	397
6.89.2 Macro Definition Documentation	397
6.89.2.1 EXECUTEPROGRAMCHECK	397
6.89.2.2 STACKCHECK	397
6.90 src/program.cpp File Reference	397
6.90.1 Detailed Description	398
6.91 src/tangBase.cpp File Reference	398
6.91.1 Detailed Description	399
6.92 src/unicodeString.cpp File Reference	399
6.92.1 Detailed Description	399
6.93 test/test.cpp File Reference	399
6.93.1 Detailed Description	401
6.94 test/testGarbageCollected.cpp File Reference	401
6.94.1 Detailed Description	402
6.95 test/testSingletonObjectPool.cpp File Reference	402
6.95.1 Detailed Description	402
6.96 test/testUnicodeString.cpp File Reference	403
6.96.1 Detailed Description	403
Index	405

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	13
Tang::AstNodeArray	18
Tang::AstNodeAssign	23
Tang::AstNodeBinary	27
Tang::AstNodeBlock	32
Tang::AstNodeBoolean	36
Tang::AstNodeBreak	40
Tang::AstNodeCast	44
Tang::AstNodeContinue	48
Tang::AstNodeDoWhile	52
Tang::AstNodeFloat	57
Tang::AstNodeFor	61
Tang::AstNodeFunctionCall	66
Tang::AstNodeFunctionDeclaration	69
Tang::AstNodeIdentifier	73
Tang::AstNodeIfElse	77
Tang::AstNodeIndex	83
Tang::AstNodeInteger	88
Tang::AstNodePrint	92
Tang::AstNodeRangedFor	96
Tang::AstNodeReturn	100
Tang::AstNodeSlice	104
Tang::AstNodeString	109
Tang::AstNodeTernary	114
Tang::AstNodeUnary	119
Tang::AstNodeWhile	124
Tang::ComputedExpression	128
Tang::ComputedExpressionArray	141
Tang::ComputedExpressionBoolean	155
Tang::ComputedExpressionCompiledFunction	167
Tang::ComputedExpressionError	180
Tang::ComputedExpressionFloat	193
Tang::ComputedExpressionInteger	208
Tang::ComputedExpressionIterator	223
Tang::ComputedExpressionIteratorEnd	237

Tang::ComputedExpressionString	249
Tang::Error	264
Tang::GarbageCollected	266
Tang::location	287
Tang::position	289
Tang::Program	290
Tang::SingletonObjectPool< T >	301
Tang::TangBase	304
TangHtmlEscapeAsciiFlexLexer	
Tang::HtmlEscapeAscii	285
TangHtmlEscapeFlexLexer	
Tang::HtmlEscape	283
TangTangFlexLexer	
Tang::TangScanner	305
TangUnescapeFlexLexer	
Tang::Unescape	308
Tang::UnicodeString	310

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)	13
Tang::AstNodeArray	An AstNode that represents an array literal	18
Tang::AstNodeAssign	An AstNode that represents a binary expression	23
Tang::AstNodeBinary	An AstNode that represents a binary expression	27
Tang::AstNodeBlock	An AstNode that represents a code block	32
Tang::AstNodeBoolean	An AstNode that represents a boolean literal	36
Tang::AstNodeBreak	An AstNode that represents a <code>break</code> statement	40
Tang::AstNodeCast	An AstNode that represents a typecast of an expression	44
Tang::AstNodeContinue	An AstNode that represents a <code>continue</code> statement	48
Tang::AstNodeDoWhile	An AstNode that represents a <code>do..while</code> statement	52
Tang::AstNodeFloat	An AstNode that represents an float literal	57
Tang::AstNodeFor	An AstNode that represents an <code>if()</code> statement	61
Tang::AstNodeFunctionCall	An AstNode that represents a function call	66
Tang::AstNodeFunctionDeclaration	An AstNode that represents a function declaration	69
Tang::AstNodeIdentifier	An AstNode that represents an identifier	73
Tang::AstNodeIfElse	An AstNode that represents an <code>if..else</code> statement	77
Tang::AstNodeIndex	An AstNode that represents an index into a collection	83
Tang::AstNodeInteger	An AstNode that represents an integer literal	88

Tang::AstNodePrint	
An AstNode that represents a print typeoperation	92
Tang::AstNodeRangedFor	
An AstNode that represents a ranged for() statement	96
Tang::AstNodeReturn	
An AstNode that represents a <code>return</code> statement	100
Tang::AstNodeSlice	
An AstNode that represents a ternary expression	104
Tang::AstNodeString	
An AstNode that represents a string literal	109
Tang::AstNodeTernary	
An AstNode that represents a ternary expression	114
Tang::AstNodeUnary	
An AstNode that represents a unary negation	119
Tang::AstNodeWhile	
An AstNode that represents a while statement	124
Tang::ComputedExpression	
Represents the result of a computation that has been executed	128
Tang::ComputedExpressionArray	
Represents an Array that is the result of a computation	141
Tang::ComputedExpressionBoolean	
Represents an Boolean that is the result of a computation	155
Tang::ComputedExpressionCompiledFunction	
Represents a Compiled Function declared in the script	167
Tang::ComputedExpressionError	
Represents a Runtime Error	180
Tang::ComputedExpressionFloat	
Represents a Float that is the result of a computation	193
Tang::ComputedExpressionInteger	
Represents an Integer that is the result of a computation	208
Tang::ComputedExpressionIterator	
Represents an Iterator that is the result of a computation	223
Tang::ComputedExpressionIteratorEnd	
Represents that a collection has no more values through which to iterate	237
Tang::ComputedExpressionString	
Represents a String that is the result of a computation	249
Tang::Error	
Used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error	264
Tang::GarbageCollected	
A container that acts as a resource-counting garbage collector for the specified type	266
Tang::HtmlEscape	
The Flex lexer class for the main Tang language	283
Tang::HtmlEscapeAscii	
The Flex lexer class for the main Tang language	285
Tang::location	
Two points in a source file	287
Tang::position	
A point in a source file	289
Tang::Program	
Represents a compiled script or template that may be executed	290
Tang::SingletonObjectPool< T >	
A thread-safe, singleton object pool of the designated type	301
Tang::TangBase	
The base class for the Tang programming language	304
Tang::TangScanner	
The Flex lexer class for the main Tang language	305

Tang::Unescape	
The Flex lexer class for the main Tang language	308
Tang::UnicodeString	
Represents a UTF-8 encoded string that is Unicode-aware	310

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	315
include/astNode.hpp	
Declare the <code>Tang::AstNode</code> base class	317
include/astNodeArray.hpp	
Declare the <code>Tang::AstNodeArray</code> class	318
include/astNodeAssign.hpp	
Declare the <code>Tang::AstNodeAssign</code> class	319
include/astNodeBinary.hpp	
Declare the <code>Tang::AstNodeBinary</code> class	320
include/astNodeBlock.hpp	
Declare the <code>Tang::AstNodeBlock</code> class	321
include/astNodeBoolean.hpp	
Declare the <code>Tang::AstNodeBoolean</code> class	322
include/astNodeBreak.hpp	
Declare the <code>Tang::AstNodeBreak</code> class	323
include/astNodeCast.hpp	
Declare the <code>Tang::AstNodeCast</code> class	324
include/astNodeContinue.hpp	
Declare the <code>Tang::AstNodeContinue</code> class	325
include/astNodeDoWhile.hpp	
Declare the <code>Tang::AstNodeDoWhile</code> class	326
include/astNodeFloat.hpp	
Declare the <code>Tang::AstNodeFloat</code> class	327
include/astNodeFor.hpp	
Declare the <code>Tang::AstNodeFor</code> class	328
include/astNodeFunctionCall.hpp	
Declare the <code>Tang::AstNodeFunctionCall</code> class	329
include/astNodeFunctionDeclaration.hpp	
Declare the <code>Tang::AstNodeFunctionDeclaration</code> class	330
include/astNodeIdentifier.hpp	
Declare the <code>Tang::AstNodeIdentifier</code> class	331
include/astNodeIfElse.hpp	
Declare the <code>Tang::AstNodeIfElse</code> class	332
include/astNodeIndex.hpp	
Declare the <code>Tang::AstNodeIndex</code> class	333

include/ astNodeInteger.hpp	Declare the Tang::AstNodeInteger class	334
include/ astNodePrint.hpp	Declare the Tang::AstNodePrint class	335
include/ astNodeRangedFor.hpp	Declare the Tang::AstNodeRangedFor class	336
include/ astNodeReturn.hpp	Declare the Tang::AstNodeReturn class	337
include/ astNodeSlice.hpp	Declare the Tang::AstNodeSlice class	338
include/ astNodeString.hpp	Declare the Tang::AstNodeString class	339
include/ astNodeTernary.hpp	Declare the Tang::AstNodeTernary class	340
include/ astNodeUnary.hpp	Declare the Tang::AstNodeUnary class	341
include/ astNodeWhile.hpp	Declare the Tang::AstNodeWhile class	342
include/ computedExpression.hpp	Declare the Tang::ComputedExpression base class	343
include/ computedExpressionArray.hpp	Declare the Tang::ComputedExpressionArray class	344
include/ computedExpressionBoolean.hpp	Declare the Tang::ComputedExpressionBoolean class	345
include/ computedExpressionCompiledFunction.hpp	Declare the Tang::ComputedExpressionCompiledFunction class	346
include/ computedExpressionError.hpp	Declare the Tang::ComputedExpressionError class	347
include/ computedExpressionFloat.hpp	Declare the Tang::ComputedExpressionFloat class	348
include/ computedExpressionInteger.hpp	Declare the Tang::ComputedExpressionInteger class	349
include/ computedExpressionIterator.hpp	Declare the Tang::ComputedExpressionIterator class	350
include/ computedExpressionIteratorEnd.hpp	Declare the Tang::ComputedExpressionIteratorEnd class	351
include/ computedExpressionString.hpp	Declare the Tang::ComputedExpressionString class	352
include/ error.hpp	Declare the Tang::Error class used to describe syntax and runtime errors	352
include/ garbageCollected.hpp	Declare the Tang::GarbageCollected class	353
include/ htmlEscape.hpp	Declare the Tang::HtmlEscape used to tokenize a Tang script	354
include/ htmlEscapeAscii.hpp	Declare the Tang::HtmlEscapeAscii used to tokenize a Tang script	356
include/ macros.hpp	Contains generic macros	357
include/ opcode.hpp	Declare the Opcodes used in the Bytecode representation of a program	357
include/ program.hpp	Declare the Tang::Program class used to compile and execute source code	359
include/ singletonObjectPool.hpp	Declare the Tang::SingletonObjectPool class	360
include/ tang.hpp	Header file supplied for use by 3rd party code so that they can easily include all necessary headers	361

include/tangBase.hpp	Declare the Tang::TangBase class used to interact with Tang	362
include/tangScanner.hpp	Declare the Tang::TangScanner used to tokenize a Tang script	363
include/unescape.hpp	Declare the Tang::Unescape used to tokenize a Tang script	364
include/unicodeString.hpp	Contains the code to interface with the ICU library	365
src/astNode.cpp	Define the Tang::AstNode class	368
src/astNodeArray.cpp	Define the Tang::AstNodeArray class	368
src/astNodeAssign.cpp	Define the Tang::AstNodeAssign class	369
src/astNodeBinary.cpp	Define the Tang::AstNodeBinary class	370
src/astNodeBlock.cpp	Define the Tang::AstNodeBlock class	371
src/astNodeBoolean.cpp	Define the Tang::AstNodeBoolean class	371
src/astNodeBreak.cpp	Define the Tang::AstNodeBreak class	372
src/astNodeCast.cpp	Define the Tang::AstNodeCast class	373
src/astNodeContinue.cpp	Define the Tang::AstNodeContinue class	373
src/astNodeDoWhile.cpp	Define the Tang::AstNodeDoWhile class	374
src/astNodeFloat.cpp	Define the Tang::AstNodeFloat class	375
src/astNodeFor.cpp	Define the Tang::AstNodeFor class	376
src/astNodeFunctionCall.cpp	Define the Tang::AstNodeFunctionCall class	376
src/astNodeFunctionDeclaration.cpp	Define the Tang::AstNodeFunctionDeclaration class	377
src/astNodeIdentifier.cpp	Define the Tang::AstNodeIdentifier class	378
src/astNodeIfElse.cpp	Define the Tang::AstNodeIfElse class	379
src/astNodeIndex.cpp	Define the Tang::AstNodeIndex class	379
src/astNodeInteger.cpp	Define the Tang::AstNodeInteger class	380
src/astNodePrint.cpp	Define the Tang::AstNodePrint class	381
src/astNodeRangedFor.cpp	Define the Tang::AstNodeRangedFor class	381
src/astNodeReturn.cpp	Define the Tang::AstNodeReturn class	382
src/astNodeSlice.cpp	Define the Tang::AstNodeSlice class	383
src/astNodeString.cpp	Define the Tang::AstNodeString class	384
src/astNodeTernary.cpp	Define the Tang::AstNodeTernary class	385
src/astNodeUnary.cpp	Define the Tang::AstNodeUnary class	386

src/ astNodeWhile.cpp	Define the Tang::AstNodeWhile class	386
src/ computedExpression.cpp	Define the Tang::ComputedExpression class	387
src/ computedExpressionArray.cpp	Define the Tang::ComputedExpressionArray class	388
src/ computedExpressionBoolean.cpp	Define the Tang::ComputedExpressionBoolean class	389
src/ computedExpressionCompiledFunction.cpp	Define the Tang::ComputedExpressionCompiledFunction class	389
src/ computedExpressionError.cpp	Define the Tang::ComputedExpressionError class	390
src/ computedExpressionFloat.cpp	Define the Tang::ComputedExpressionFloat class	391
src/ computedExpressionInteger.cpp	Define the Tang::ComputedExpressionInteger class	391
src/ computedExpressionIterator.cpp	Define the Tang::ComputedExpressionIterator class	392
src/ computedExpressionIteratorEnd.cpp	Define the Tang::ComputedExpressionIteratorEnd class	393
src/ computedExpressionString.cpp	Define the Tang::ComputedExpressionString class	393
src/ error.cpp	Define the Tang::Error class	394
src/ program-dumpBytecode.cpp	Define the Tang::Program::dumpBytecode method	395
src/ program-execute.cpp	Define the Tang::Program::execute method	396
src/ program.cpp	Define the Tang::Program class	397
src/ tangBase.cpp	Define the Tang::TangBase class	398
src/ unicodeString.cpp	Contains the function declarations for the Tang::UnicodeString class and the interface to ICU	399
test/ test.cpp	Test the general language behaviors	399
test/ testGarbageCollected.cpp	Test the generic behavior of the Tang::GarbageCollected class	401
test/ testSingletonObjectPool.cpp	Test the generic behavior of the Tang::SingletonObjectPool class	402
test/ testUnicodeString.cpp	Contains tests for the Tang::UnicodeString class	403

Chapter 5

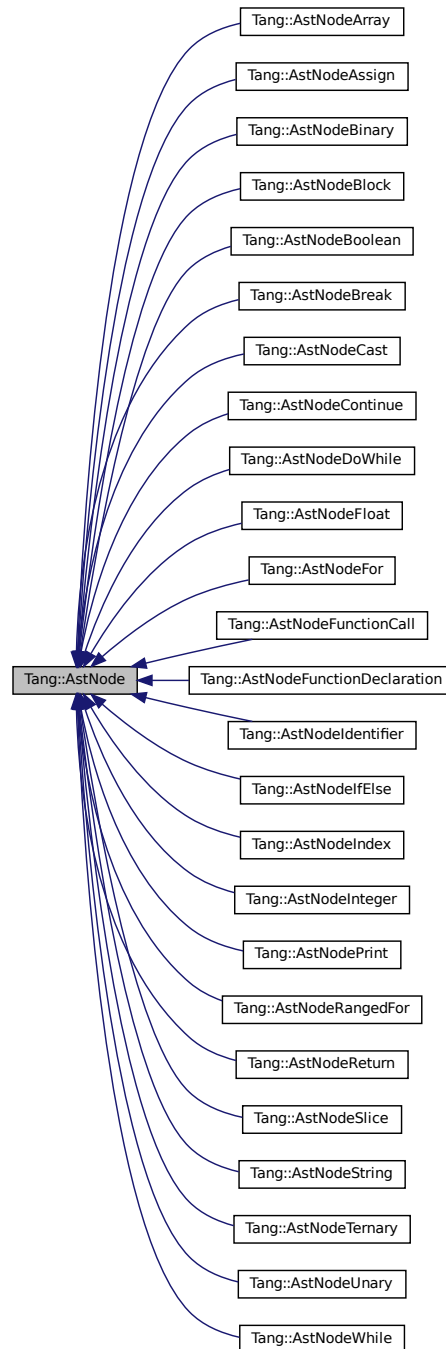
Class Documentation

5.1 Tang::AstNode Class Reference

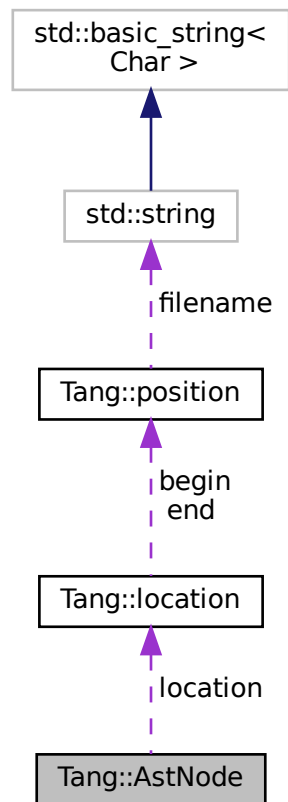
Base class for representing nodes of an Abstract Syntax Tree (AST).

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

Inheritance diagram for Tang::AstNode:



Collaboration diagram for Tang::AstNode:



Public Types

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

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

Public Member Functions

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

Private Attributes

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

The location associated with this node.

5.1.1 Detailed Description

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

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

5.1.2 Member Enumeration Documentation

5.1.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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::AstNodeRangedFor](#), [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::AstNodeRangedFor](#), [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::AstNodeRangedFor](#), [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:

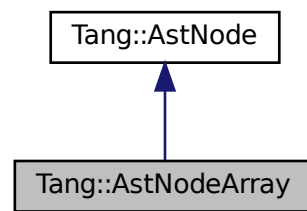
- [include/astNode.hpp](#)
- [src/astNode.cpp](#)

5.2 Tang::AstNodeArray Class Reference

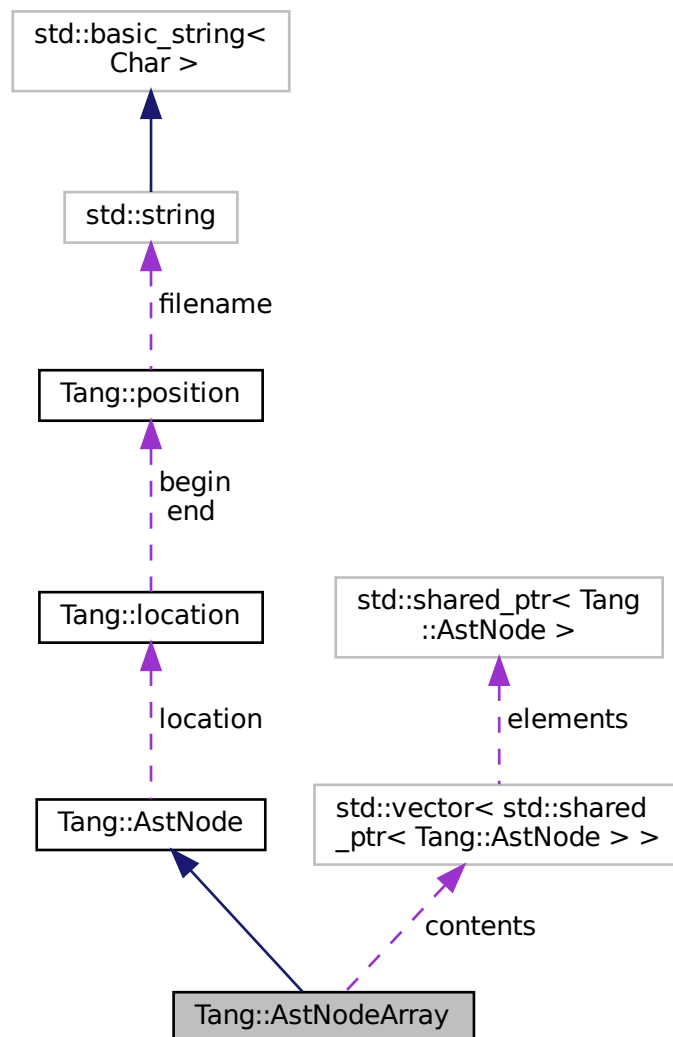
An [AstNode](#) that represents an array literal.

```
#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.

Private Attributes

- `std::vector< std::shared_ptr< Tang::AstNode > > contents`
The contents of the array.
- `Tang::location location`
The location associated with this node.

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]
```

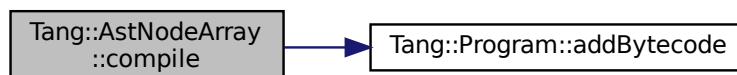
Compile the ast of the provided [Tang::Program](#).

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:

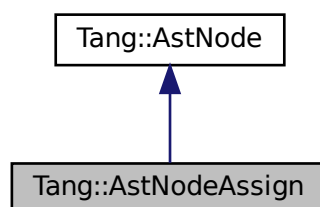
- [include/astNodeArray.hpp](#)
- [src/astNodeArray.cpp](#)

5.3 Tang::AstNodeAssign Class Reference

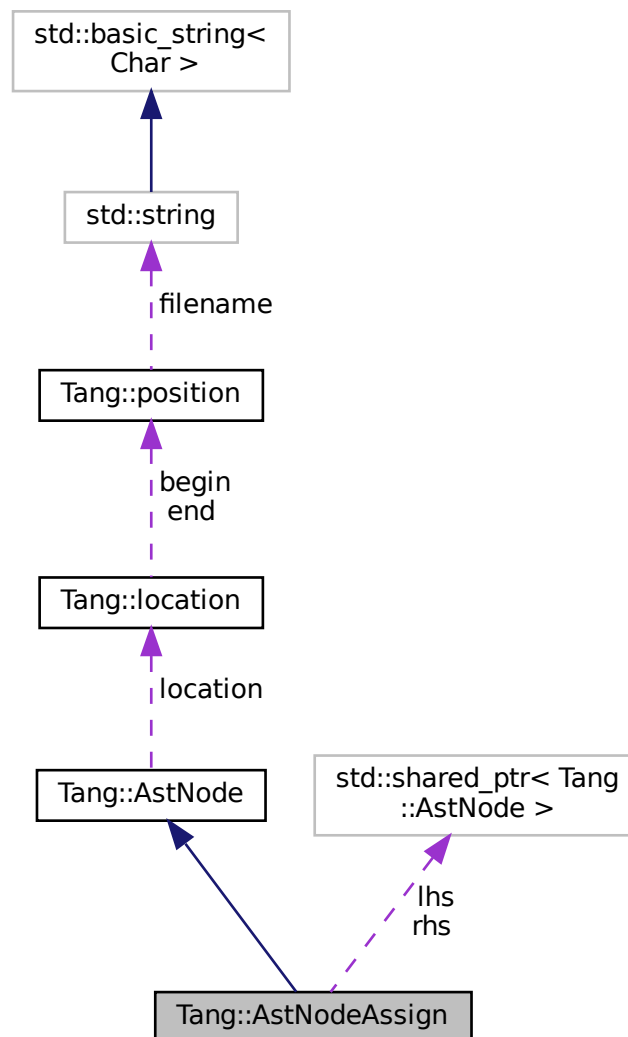
An [AstNode](#) that represents a binary expression.

```
#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.

Private Attributes

- std::shared_ptr< [AstNode](#) > [lhs](#)
The left hand side expression.
- std::shared_ptr< [AstNode](#) > [rhs](#)
The right hand side expression.
- [Tang::location](#) [location](#)
The location associated with this node.

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]
```

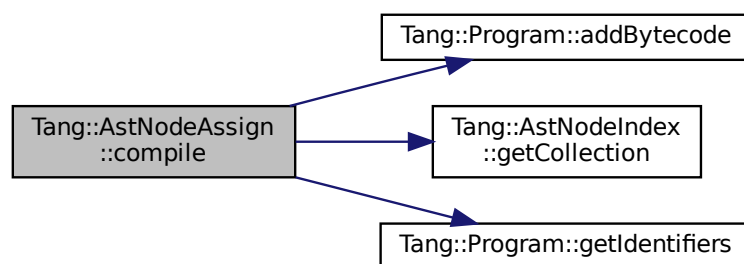
Compile the ast of the provided [Tang::Program](#).

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:

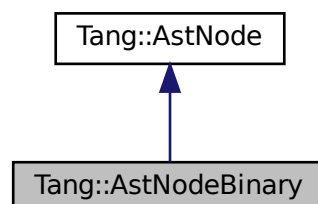
- [include/astNodeAssign.hpp](#)
- [src/astNodeAssign.cpp](#)

5.4 Tang::AstNodeBinary Class Reference

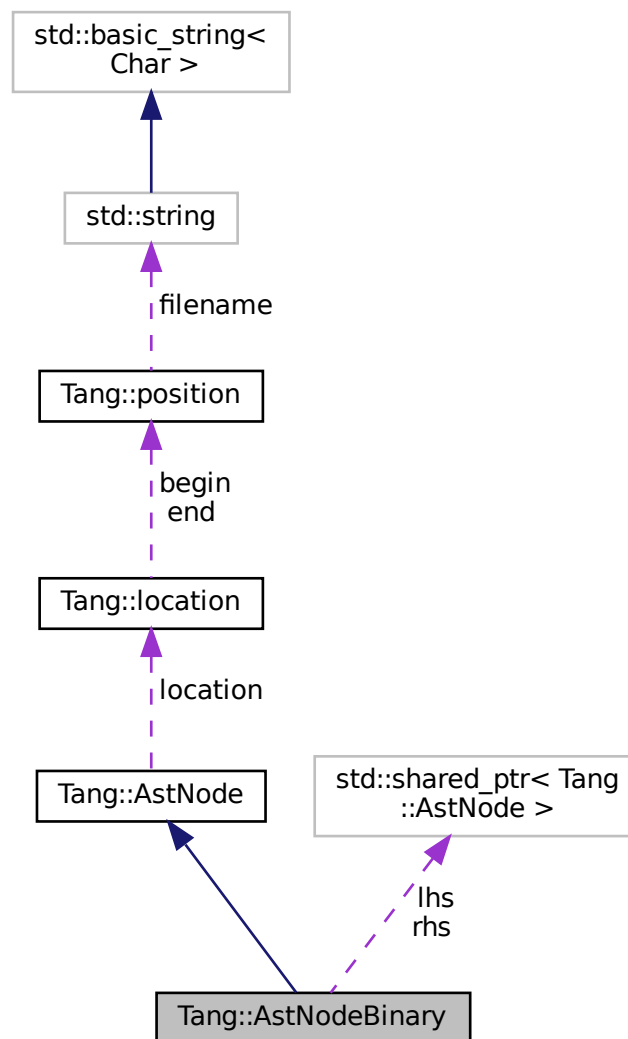
An [AstNode](#) that represents a binary expression.

```
#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.

Private Attributes

- [Operation](#) op
The binary operation performed.
- std::shared_ptr< [AstNode](#) > lhs
The left hand side expression.
- std::shared_ptr< [AstNode](#) > rhs
The right hand side expression.
- [Tang::location](#) location
The location associated with this node.

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]
```

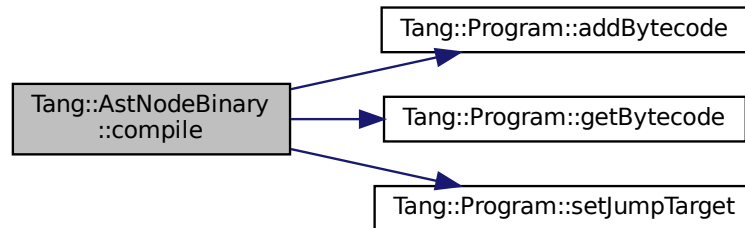
Compile the ast of the provided [Tang::Program](#).

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:

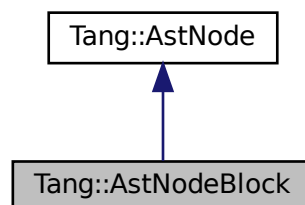
- include/[astNodeBinary.hpp](#)
- src/[astNodeBinary.cpp](#)

5.5 Tang::AstNodeBlock Class Reference

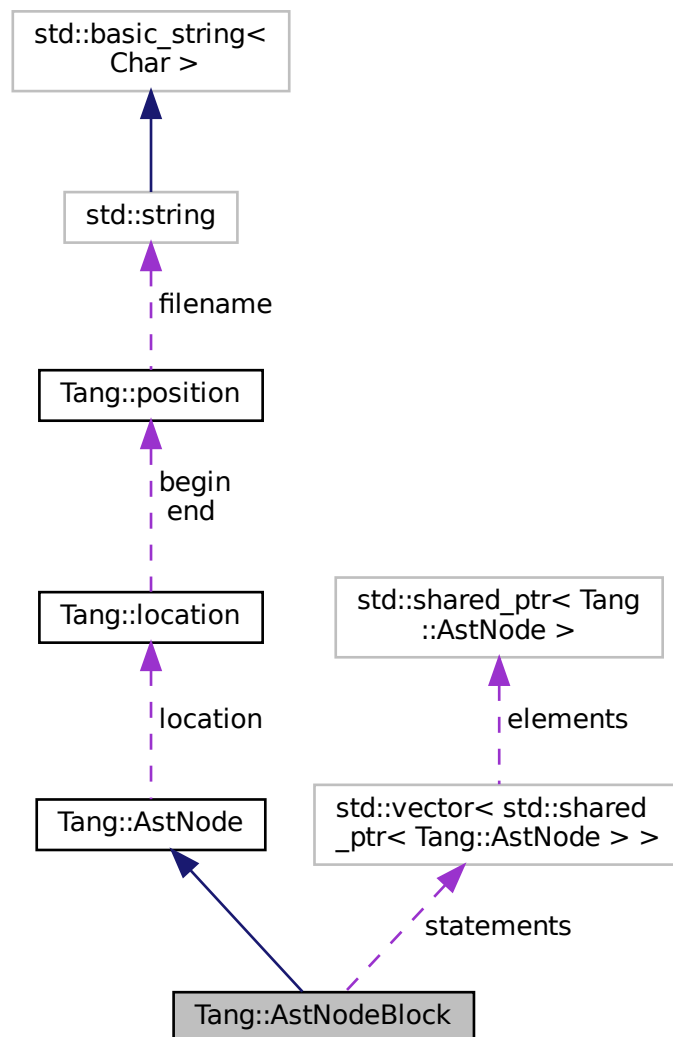
An [AstNode](#) that represents a code block.

```
#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.

Private Attributes

- `std::vector< std::shared_ptr< AstNode > > statements`

The statements included in the code block.

- `Tang::location location`

The location associated with this node.

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]
```

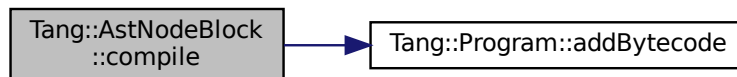
Compile the ast of the provided [Tang::Program](#).

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:

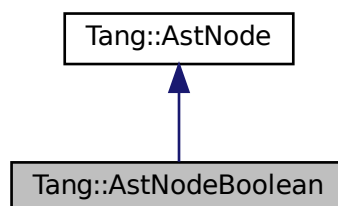
- [include/astNodeBlock.hpp](#)
- [src/astNodeBlock.cpp](#)

5.6 Tang::AstNodeBoolean Class Reference

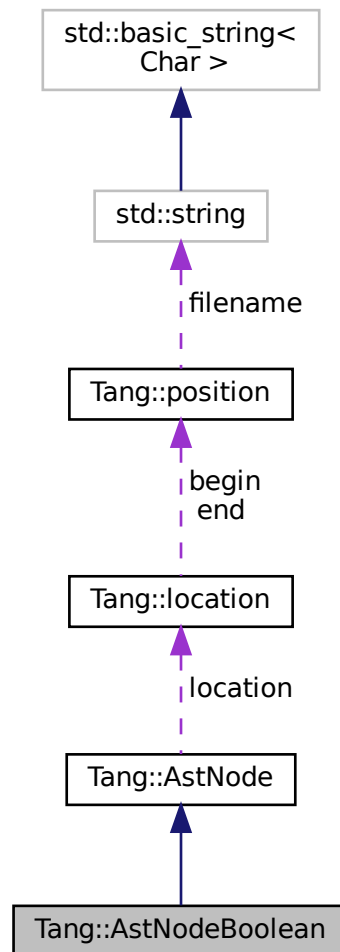
An [AstNode](#) that represents a boolean literal.

```
#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.

Private Attributes

- `bool val`
The boolean value being stored.
- `Tang::location location`
The location associated with this node.

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

5.6.3 Constructor & Destructor Documentation

5.6.3.1 AstNodeBoolean()

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

The constructor.

Parameters

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

5.6.4 Member Function Documentation

5.6.4.1 compile()

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

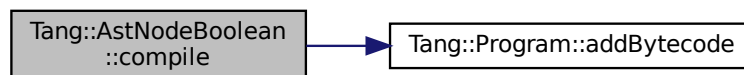
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.6.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

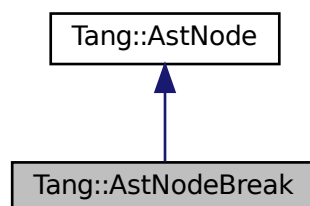
- include/[astNodeBoolean.hpp](#)
- src/[astNodeBoolean.cpp](#)

5.7 Tang::AstNodeBreak Class Reference

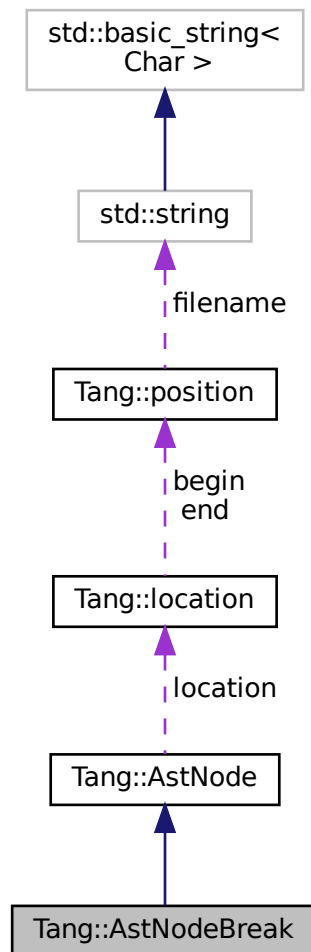
An [AstNode](#) that represents a `break` statement.

```
#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.

Private Attributes

- [Tang::location location](#)

The location associated with this node.

5.7.1 Detailed Description

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

5.7.2 Member Enumeration Documentation

5.7.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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]
```

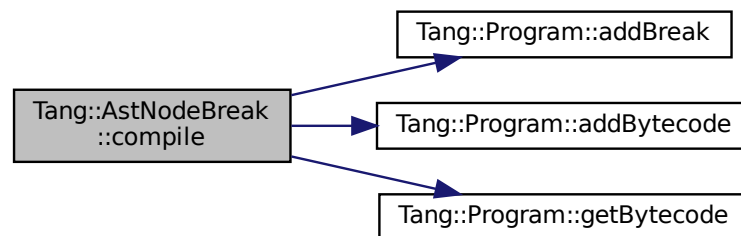
Compile the ast of the provided [Tang::Program](#).

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::AstNodeRangedFor](#), [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:

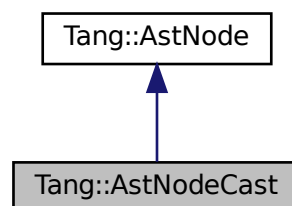
- [include/astNodeBreak.hpp](#)
- [src/astNodeBreak.cpp](#)

5.8 Tang::AstNodeCast Class Reference

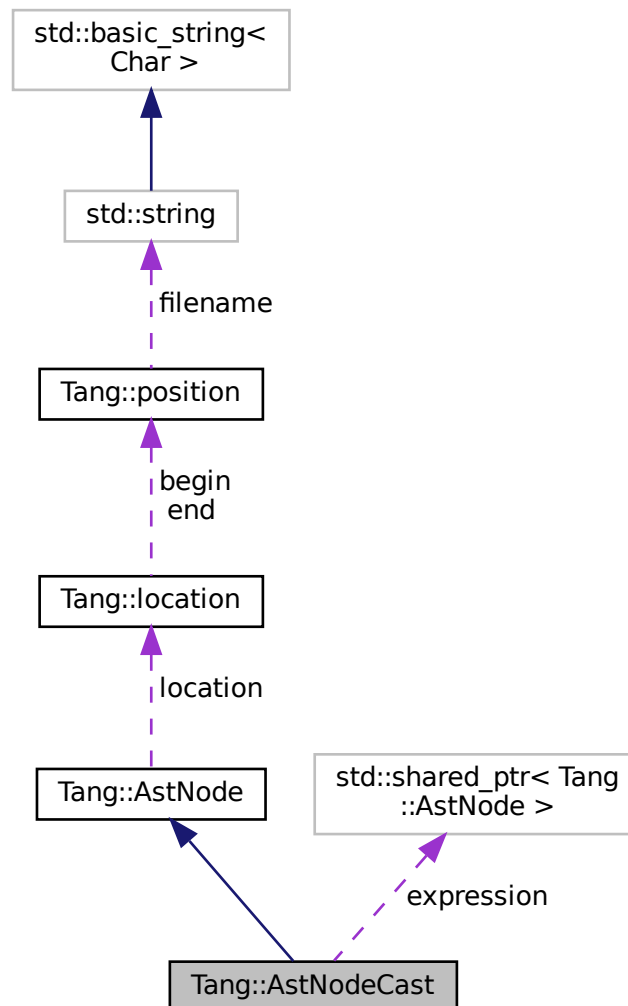
An [AstNode](#) that represents a typecast of an expression.

```
#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.

Private Attributes

- [Type](#) `targetType`

The target type.

- `shared_ptr`< [AstNode](#) > `expression`

The expression being typecast.

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

The location associated with this node.

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]
```

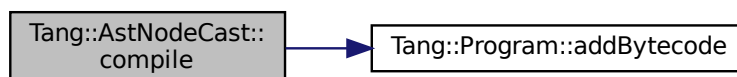
Compile the ast of the provided [Tang::Program](#).

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:

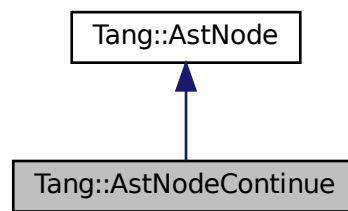
- [include/astNodeCast.hpp](#)
- [src/astNodeCast.cpp](#)

5.9 Tang::AstNodeContinue Class Reference

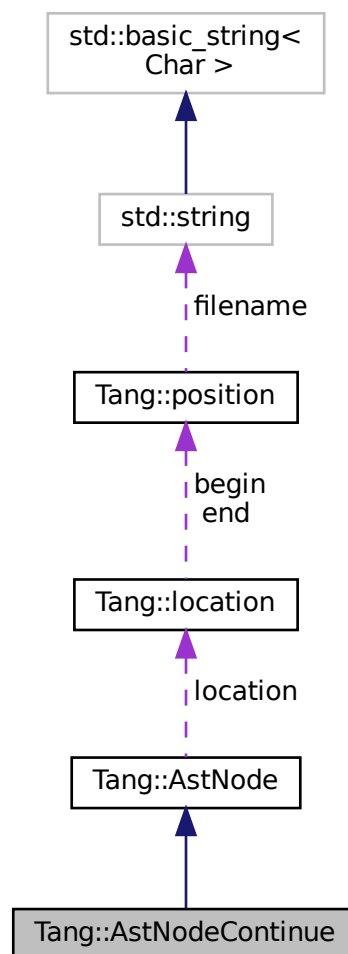
An [AstNode](#) that represents a `continue` statement.

```
#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.

Private Attributes

- [Tang::location](#) location

The location associated with this node.

5.9.1 Detailed Description

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

5.9.2 Member Enumeration Documentation

5.9.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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]
```

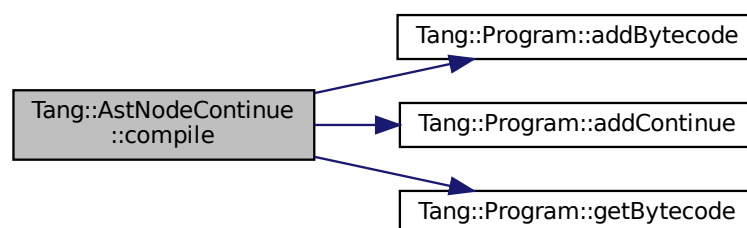
Compile the ast of the provided [Tang::Program](#).

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::AstNodeRangedFor](#), [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:

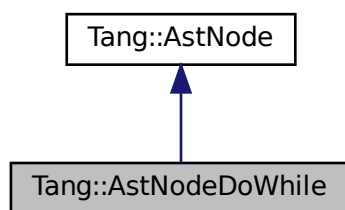
- include/[astNodeContinue.hpp](#)
- src/[astNodeContinue.cpp](#)

5.10 Tang::AstNodeDoWhile Class Reference

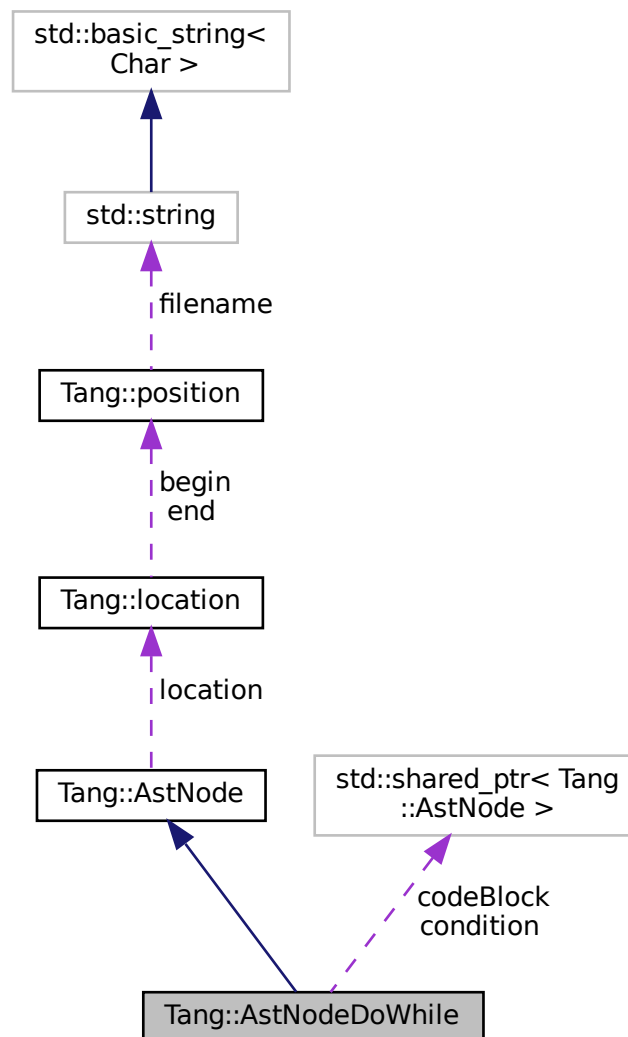
An [AstNode](#) that represents a do..while statement.

```
#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.

Private Attributes

- shared_ptr< [AstNode](#) > `condition`
The expression which determines whether or not the code block will continue to be executed.
- shared_ptr< [AstNode](#) > `codeBlock`
The code block executed when the condition is true.
- [Tang::location](#) `location`
The location associated with this node.

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]
```

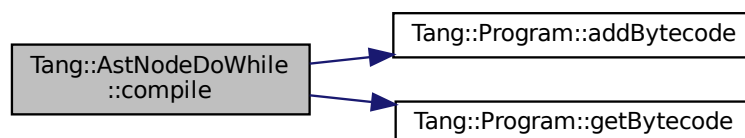
Compile the ast of the provided [Tang::Program](#).

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:

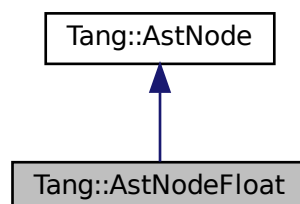
- [include/astNodeDoWhile.hpp](#)
- [src/astNodeDoWhile.cpp](#)

5.11 Tang::AstNodeFloat Class Reference

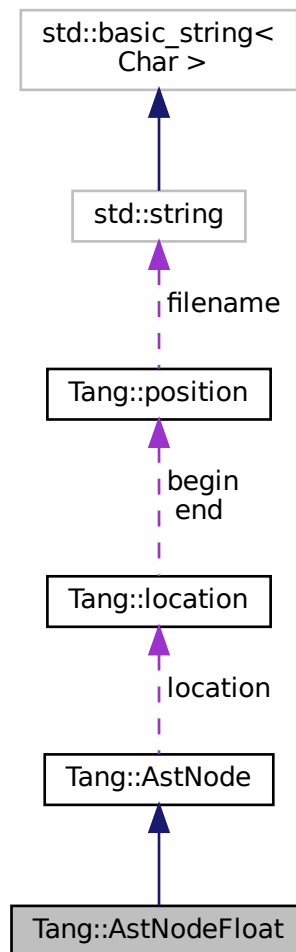
An [AstNode](#) that represents an float literal.

```
#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.

Private Attributes

- [Tang::float_t val](#)
The float value being stored.
- [Tang::location location](#)
The location associated with this node.

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]
```

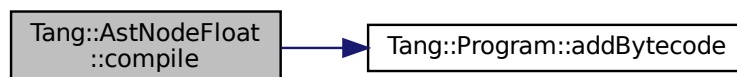
Compile the ast of the provided [Tang::Program](#).

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::AstNodeRangedFor](#), [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:

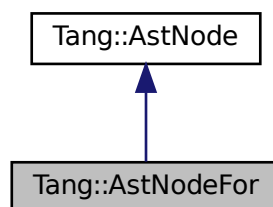
- [include/astNodeFloat.hpp](#)
- [src/astNodeFloat.cpp](#)

5.12 Tang::AstNodeFor Class Reference

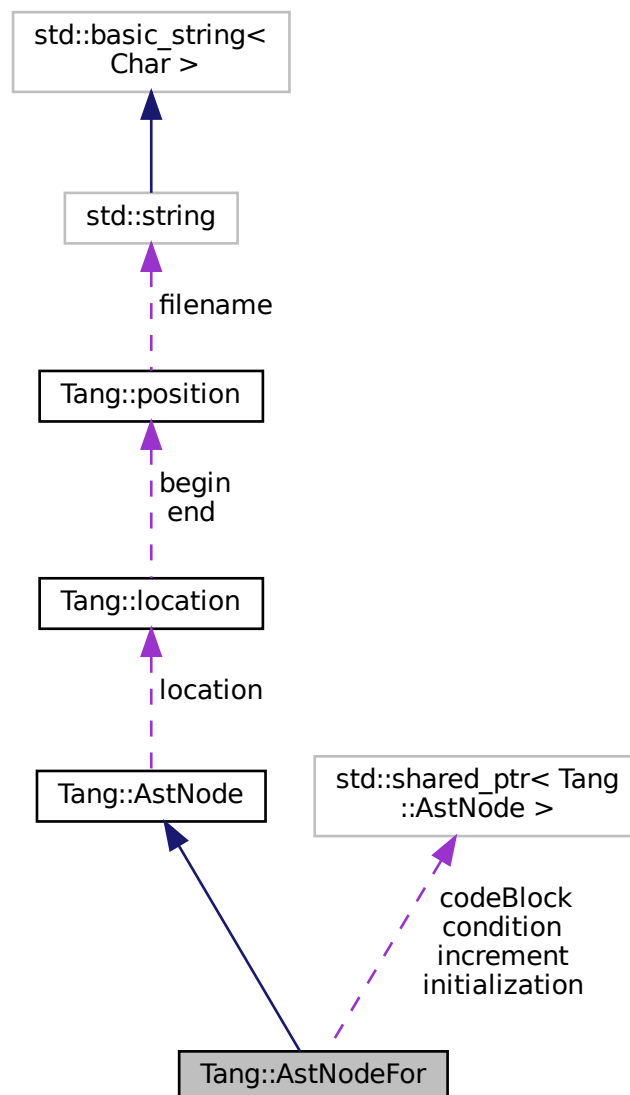
An [AstNode](#) that represents an if() statement.

```
#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.

Private Attributes

- shared_ptr< [AstNode](#) > [initialization](#)
The expression to be executed first to set up the for() loop.
- shared_ptr< [AstNode](#) > [condition](#)
The expression which determines whether or not the code block will continue to be executed.
- shared_ptr< [AstNode](#) > [increment](#)
The expression to be executed immediately after the code block.
- shared_ptr< [AstNode](#) > [codeBlock](#)
The code block executed when the condition is true.
- [Tang::location](#) [location](#)
The location associated with this node.

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]
```

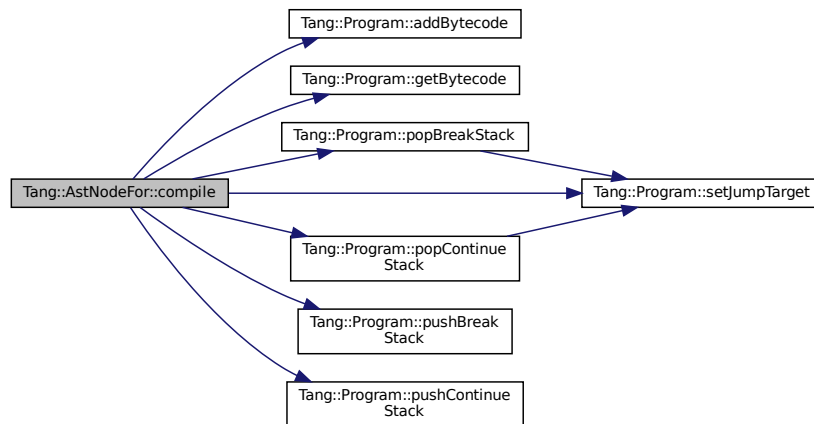
Compile the ast of the provided [Tang::Program](#).

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:

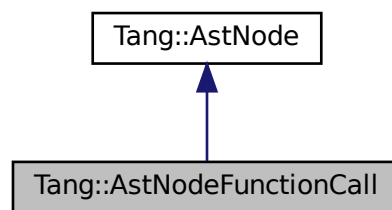
- [include/astNodeFor.hpp](#)
- [src/astNodeFor.cpp](#)

5.13 Tang::AstNodeFunctionCall Class Reference

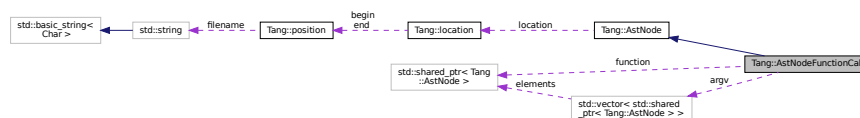
An [AstNode](#) that represents a function call.

```
#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.

Private Attributes

- std::shared_ptr< [AstNode](#) > [function](#)
The function being invoked.
- std::vector< std::shared_ptr< [AstNode](#) > > [argv](#)
The list of arguments provided to the function.
- [Tang::location](#) [location](#)
The location associated with this node.

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:

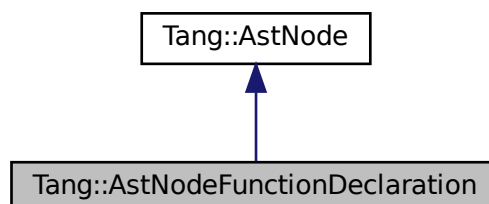
- [include/astNodeFunctionCall.hpp](#)
- [src/astNodeFunctionCall.cpp](#)

5.14 Tang::AstNodeFunctionDeclaration Class Reference

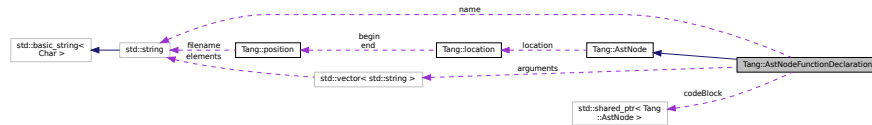
An [AstNode](#) that represents a function declaration.

```
#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.

Private Attributes

- std::string [name](#)
The name of the function.
- std::vector< std::string > [arguments](#)
The arguments expected to be provided.
- shared_ptr< [AstNode](#) > [codeBlock](#)
The code block executed when the condition is true.
- [Tang::location](#) [location](#)
The location associated with this node.

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]
```

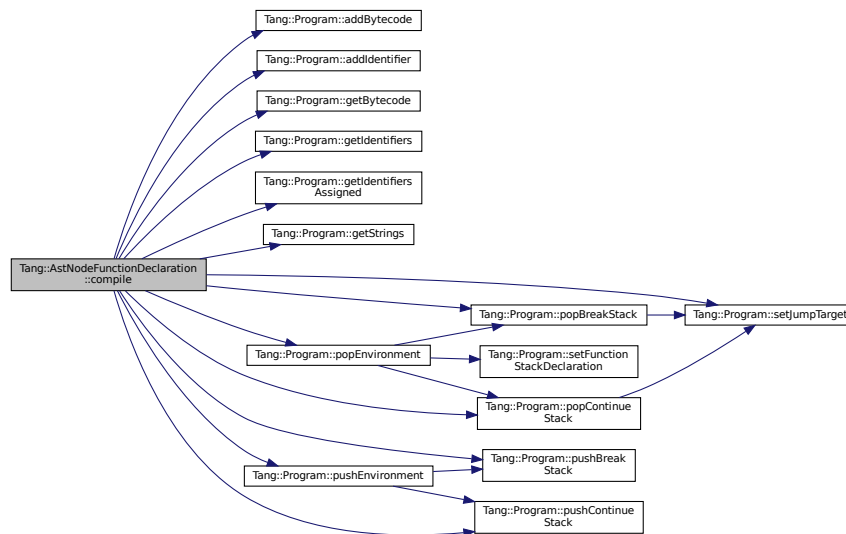
Compile the ast of the provided [Tang::Program](#).

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]

```

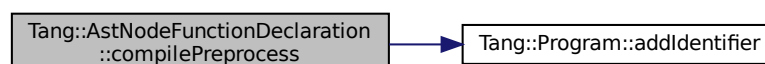
Run any preprocess analysis needed before compilation.

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:

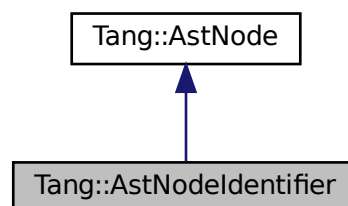
- [include/astNodeFunctionDeclaration.hpp](#)
- [src/astNodeFunctionDeclaration.cpp](#)

5.15 Tang::AstNodeIdentifier Class Reference

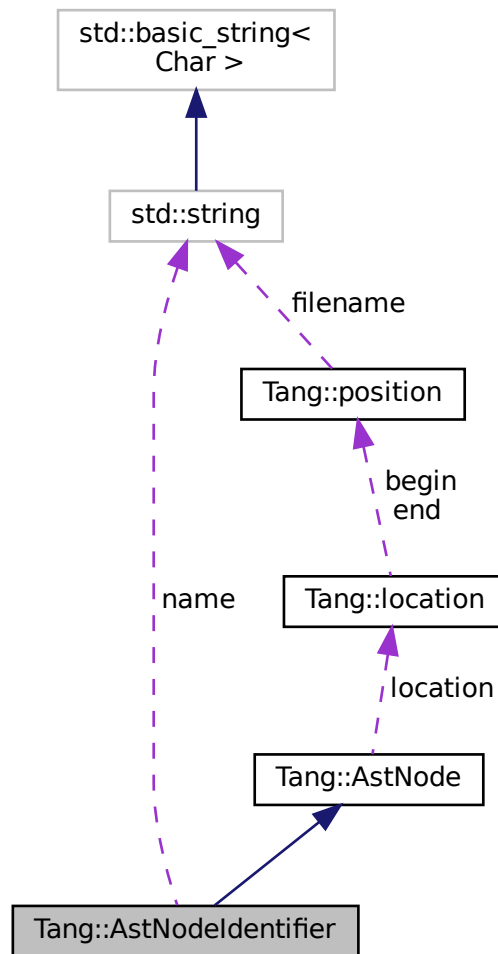
An [AstNode](#) that represents an identifier.

```
#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.

Private Attributes

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

5.15.1 Detailed Description

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

Identifier names are represented by a string.

5.15.2 Member Enumeration Documentation

5.15.2.1 PreprocessState

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

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

Enumerator

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]
```

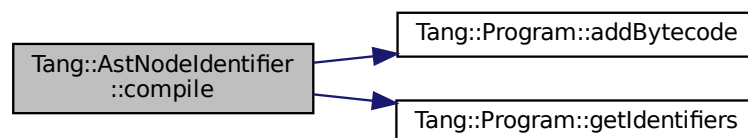
Compile the ast of the provided [Tang::Program](#).

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]
```

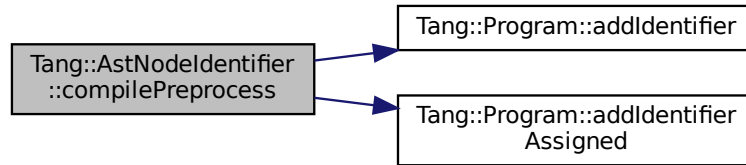
Run any preprocess analysis needed before compilation.

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:

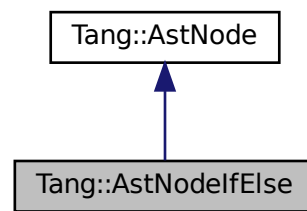
- [include/astNodeIdentifier.hpp](#)
- [src/astNodeIdentifier.cpp](#)

5.16 Tang::AstNodeIfElse Class Reference

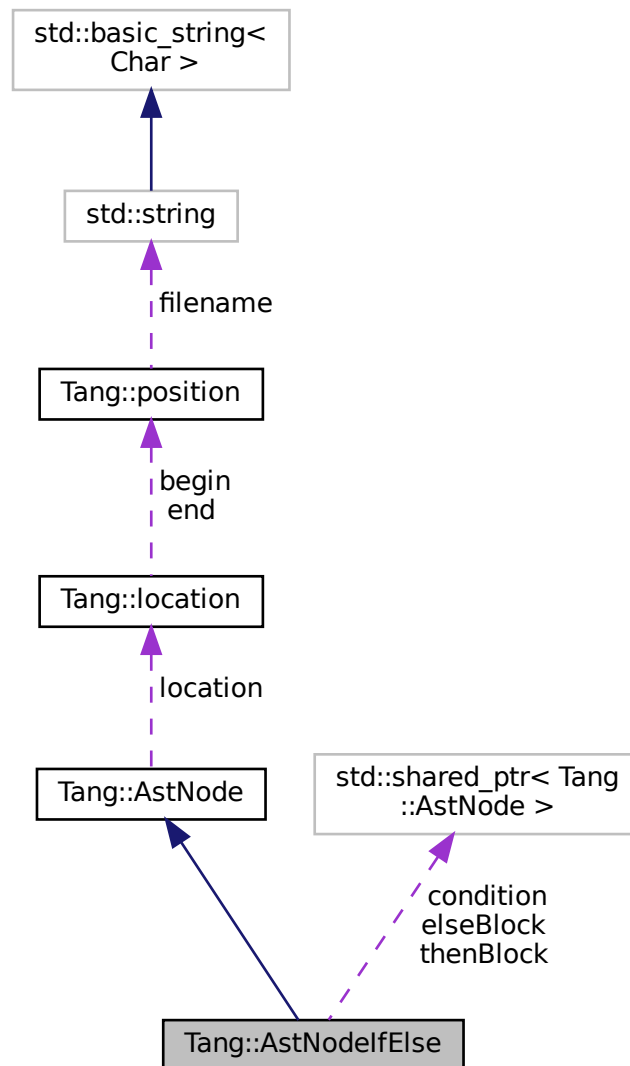
An [AstNode](#) that represents an if..else statement.

```
#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.

Private Attributes

- shared_ptr< [AstNode](#) > condition
The expression which determines whether the thenBlock or elseBlock is executed.
- shared_ptr< [AstNode](#) > thenBlock
The statement executed when the condition is true.
- shared_ptr< [AstNode](#) > elseBlock
The statement executed when the condition is false.
- [Tang::location](#) location
The location associated with this node.

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]
```

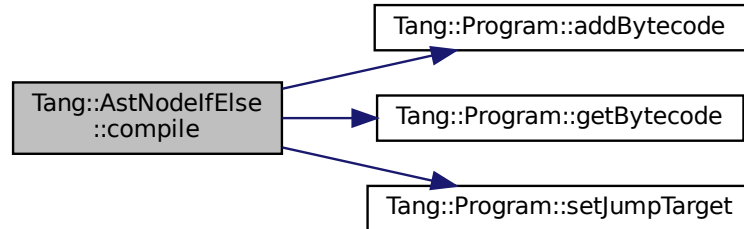
Compile the ast of the provided [Tang::Program](#).

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:

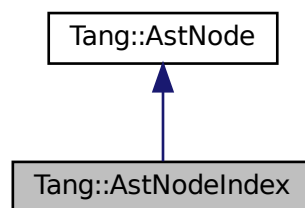
- [include/astNodeIfElse.hpp](#)
- [src/astNodeIfElse.cpp](#)

5.17 Tang::AstNodeIndex Class Reference

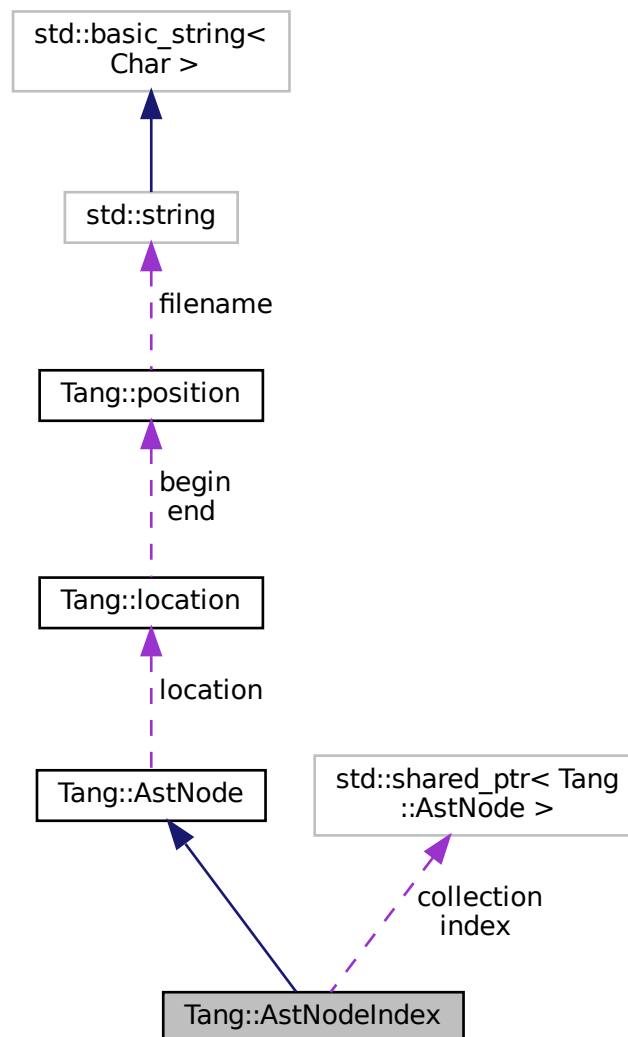
An [AstNode](#) that represents an index into a collection.

```
#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.

Private Attributes

- std::shared_ptr< [AstNode](#) > `collection`
The collection into which we will index.
- std::shared_ptr< [AstNode](#) > `index`
The index expression.
- [Tang::location](#) `location`
The location associated with this node.

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]
```

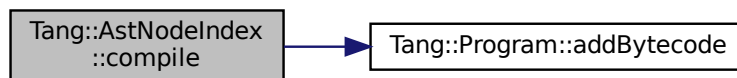
Compile the ast of the provided [Tang::Program](#).

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:

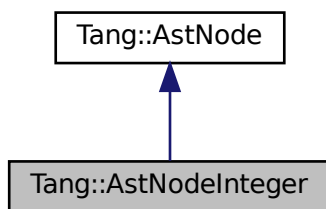
- [include/astNodeIndex.hpp](#)
- [src/astNodeIndex.cpp](#)

5.18 Tang::AstNodeInteger Class Reference

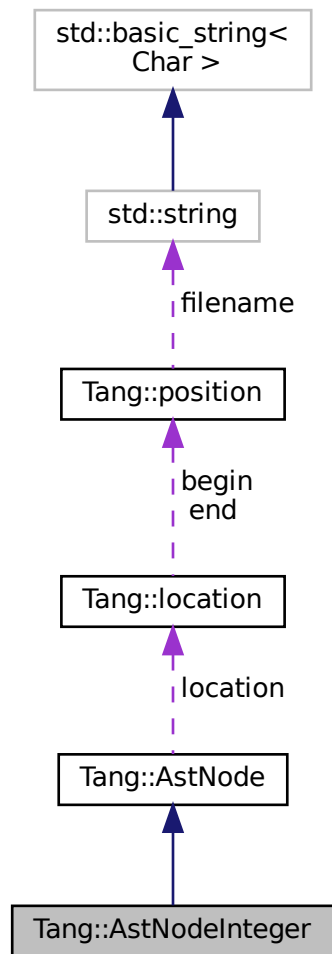
An [AstNode](#) that represents an integer literal.

```
#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.

Private Attributes

- [Tang::integer_t val](#)
The integer value being stored.
- [Tang::location location](#)
The location associated with this node.

5.18.1 Detailed Description

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

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

5.18.2 Member Enumeration Documentation

5.18.2.1 PreprocessState

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

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

Enumerator

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

5.18.3 Constructor & Destructor Documentation

5.18.3.1 AstNodeInteger()

```
AstNodeInteger::AstNodeInteger (
    Tang::integer\_t number,
    Tang::location location )
```

The constructor.

Parameters

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

5.18.4 Member Function Documentation

5.18.4.1 compile()

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

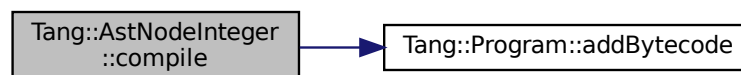
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.18.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

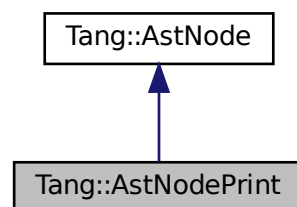
- [include/astNodeInteger.hpp](#)
- [src/astNodeInteger.cpp](#)

5.19 Tang::AstNodePrint Class Reference

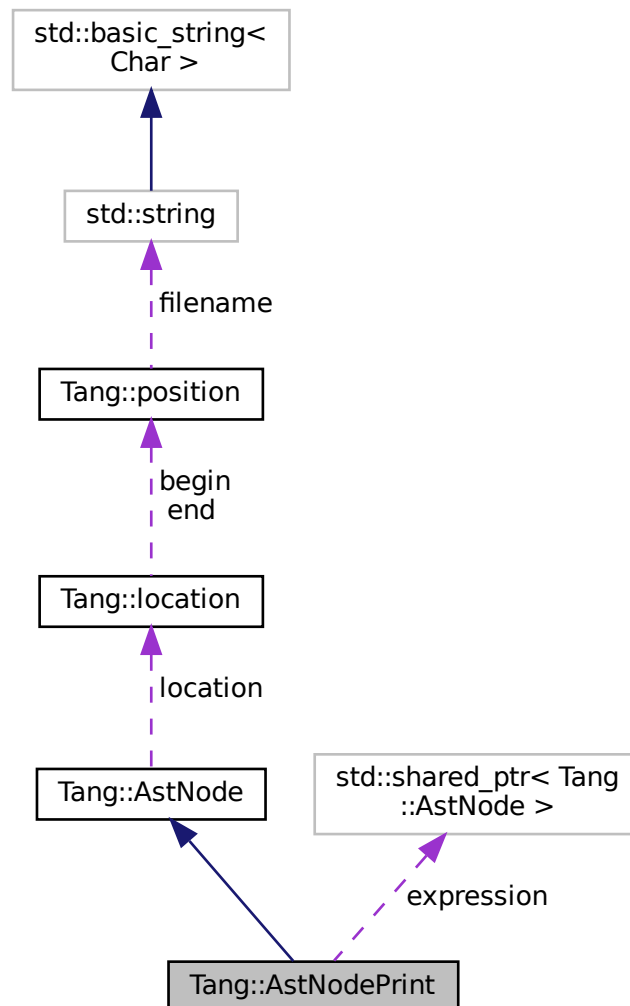
An [AstNode](#) that represents a print typeoperation.

```
#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.

Private Attributes

- [Type](#) type

The type of print() being requested.

- shared_ptr< [AstNode](#) > expression

The expression to be printed.

- [Tang::location](#) location

The location associated with this node.

5.19.1 Detailed Description

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

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]
```

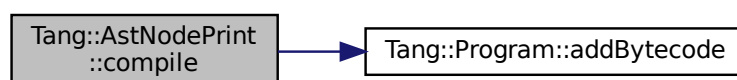
Compile the ast of the provided [Tang::Program](#).

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:

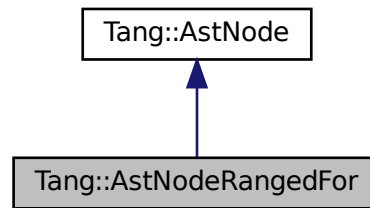
- [include/astNodePrint.hpp](#)
- [src/astNodePrint.cpp](#)

5.20 Tang::AstNodeRangedFor Class Reference

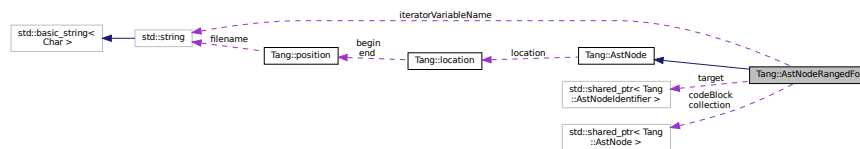
An [AstNode](#) that represents a ranged for() statement.

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

Inheritance diagram for Tang::AstNodeRangedFor:



Collaboration diagram for Tang::AstNodeRangedFor:



Public Types

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

Public Member Functions

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

Private Attributes

- shared_ptr< [AstNodeIdentifier](#) > [target](#)
- The target variable to hold the value for the current loop iteration.*
- shared_ptr< [AstNode](#) > [collection](#)
- The collection through which to iterate.*
- shared_ptr< [AstNode](#) > [codeBlock](#)
- The code block executed when the condition is true.*
- string [iteratorVariableName](#)
- The unique variable name that this iterator will use to persist its state on the stack.*
- [Tang::location](#) [location](#)
- The location associated with this node.*

5.20.1 Detailed Description

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

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

The constructor.

Parameters

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

5.20.4 Member Function Documentation

5.20.4.1 compile()

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

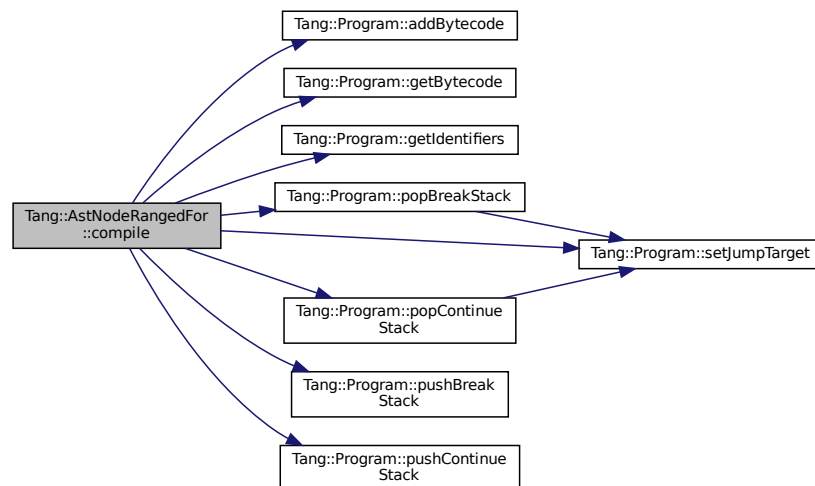
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.20.4.2 compilePreprocess()

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

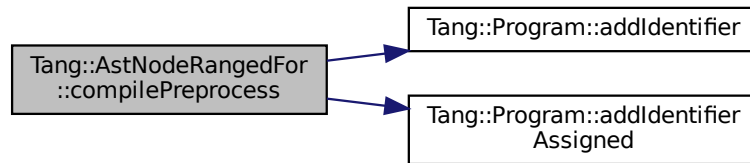
Run any preprocess analysis needed before compilation.

Parameters

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

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

Here is the call graph for this function:



5.20.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

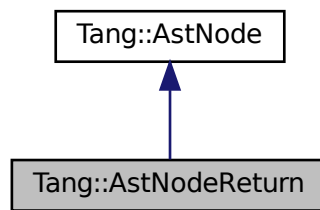
- [include/astNodeRangedFor.hpp](#)
- [src/astNodeRangedFor.cpp](#)

5.21 Tang::AstNodeReturn Class Reference

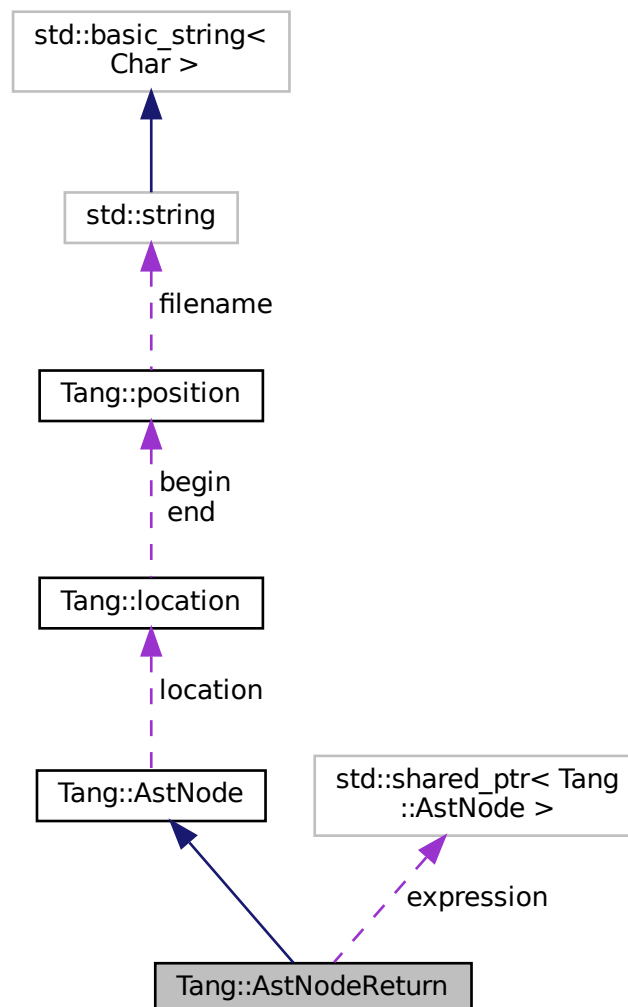
An [AstNode](#) that represents a `return` statement.

```
#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.

Private Attributes

- shared_ptr< [AstNode](#) > [expression](#)
The expression to which the operation will be applied.
- [Tang::location](#) [location](#)
The location associated with this node.

5.21.1 Detailed Description

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

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 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.

5.21.4 Member Function Documentation

5.21.4.1 compile()

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

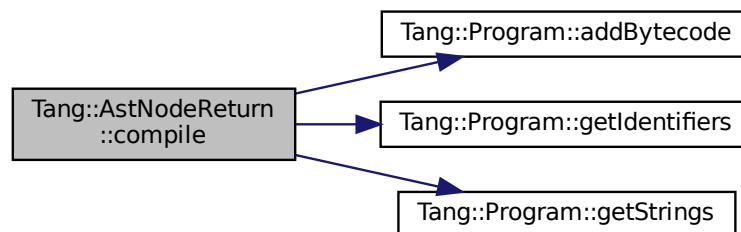
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.21.4.2 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](#).

5.21.4.3 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:

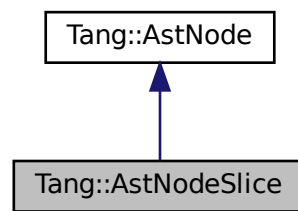
- [include/astNodeReturn.hpp](#)
- [src/astNodeReturn.cpp](#)

5.22 Tang::AstNodeSlice Class Reference

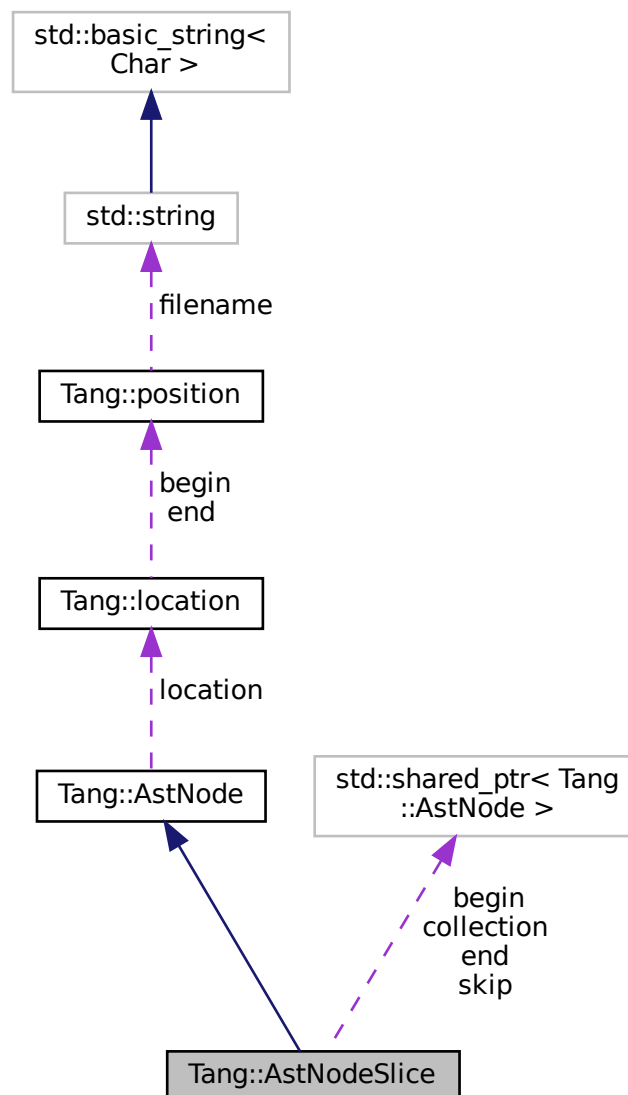
An [AstNode](#) that represents a ternary expression.

```
#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.

Private Attributes

- shared_ptr< [AstNode](#) > [collection](#)
The collection which will be sliced.
- shared_ptr< [AstNode](#) > [begin](#)
The begin index position of the slice.
- shared_ptr< [AstNode](#) > [end](#)
The end index position of the slice.
- shared_ptr< [AstNode](#) > [skip](#)
The skip index position of the slice.
- [Tang::location](#) [location](#)
The location associated with this node.

5.22.1 Detailed Description

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

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 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.

5.22.4 Member Function Documentation

5.22.4.1 compile()

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

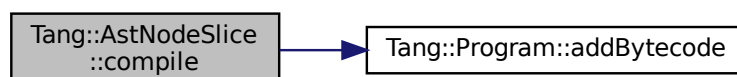
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.22.4.2 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](#).

5.22.4.3 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:

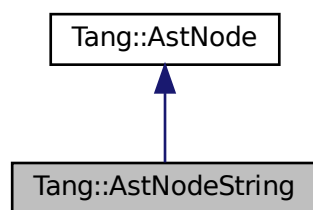
- [include/astNodeSlice.hpp](#)
- [src/astNodeSlice.cpp](#)

5.23 Tang::AstNodeString Class Reference

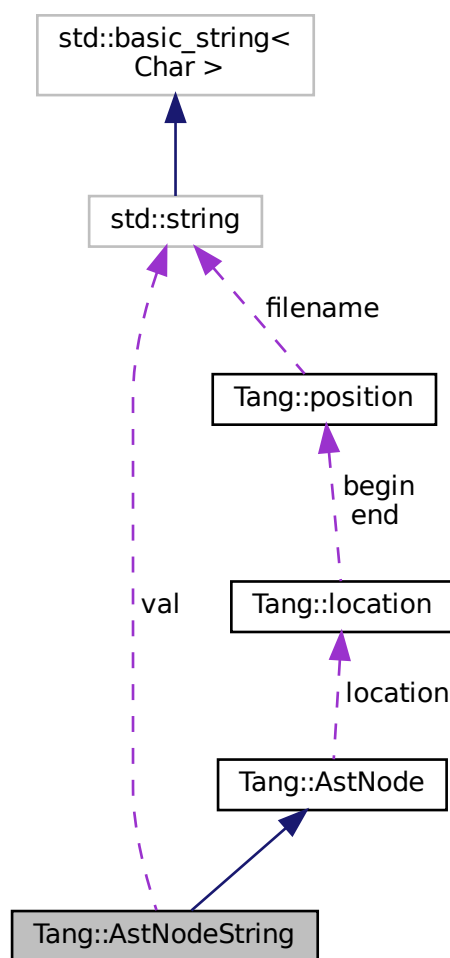
An [AstNode](#) that represents a string literal.

```
#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.

Private Attributes

- std::string [val](#)
The string value being stored.
- [Tang::location](#) [location](#)
The location associated with this node.

5.23.1 Detailed Description

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

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 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.

5.23.4 Member Function Documentation

5.23.4.1 compile()

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

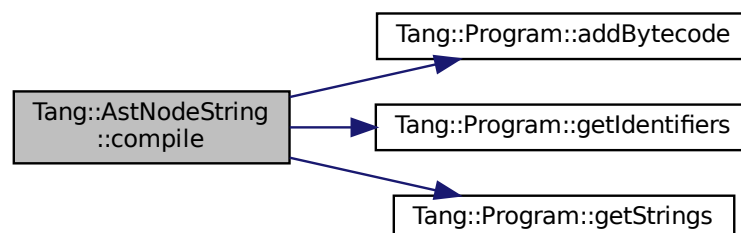
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.23.4.2 compileLiteral()

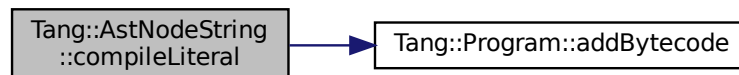
```
void AstNodeString::compileLiteral (
    Tang::Program & program ) const
```

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:



5.23.4.3 compilePreprocess()

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

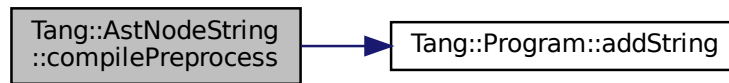
Run any preprocess analysis needed before compilation.

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:



5.23.4.4 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:

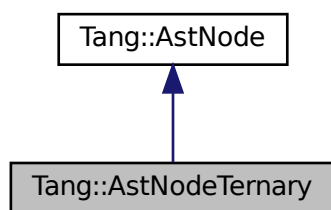
- [include/astNodeString.hpp](#)
- [src/astNodeString.cpp](#)

5.24 Tang::AstNodeTernary Class Reference

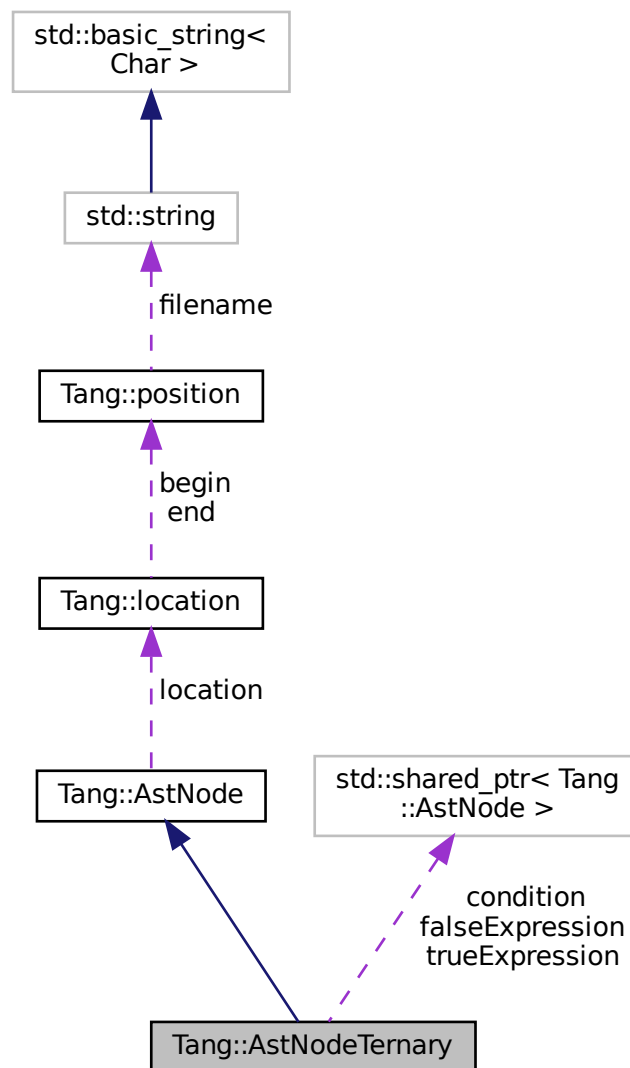
An [AstNode](#) that represents a ternary expression.

```
#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.

Private Attributes

- shared_ptr< [AstNode](#) > `condition`

The expression which determines whether the trueExpression or falseExpression is executed.

- shared_ptr< [AstNode](#) > `trueExpression`

The expression executed when the condition is true.

- shared_ptr< [AstNode](#) > `falseExpression`

The expression executed when the condition is false.

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

The location associated with this node.

5.24.1 Detailed Description

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

5.24.2 Member Enumeration Documentation

5.24.2.1 PreprocessState

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

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

Enumerator

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

5.24.3 Constructor & Destructor Documentation

5.24.3.1 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.

5.24.4 Member Function Documentation

5.24.4.1 compile()

```

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

```

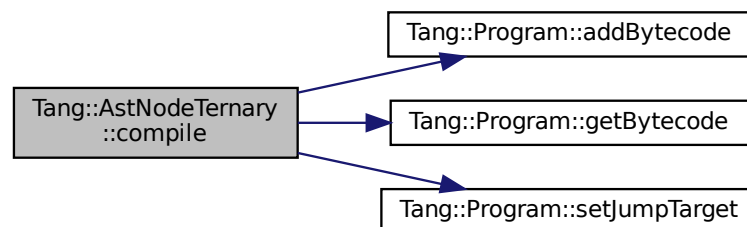
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.24.4.2 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](#).

5.24.4.3 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:

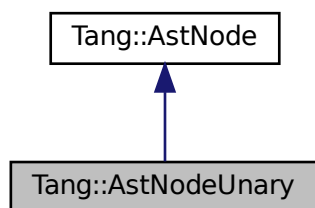
- [include/astNodeTernary.hpp](#)
- [src/astNodeTernary.cpp](#)

5.25 Tang::AstNodeUnary Class Reference

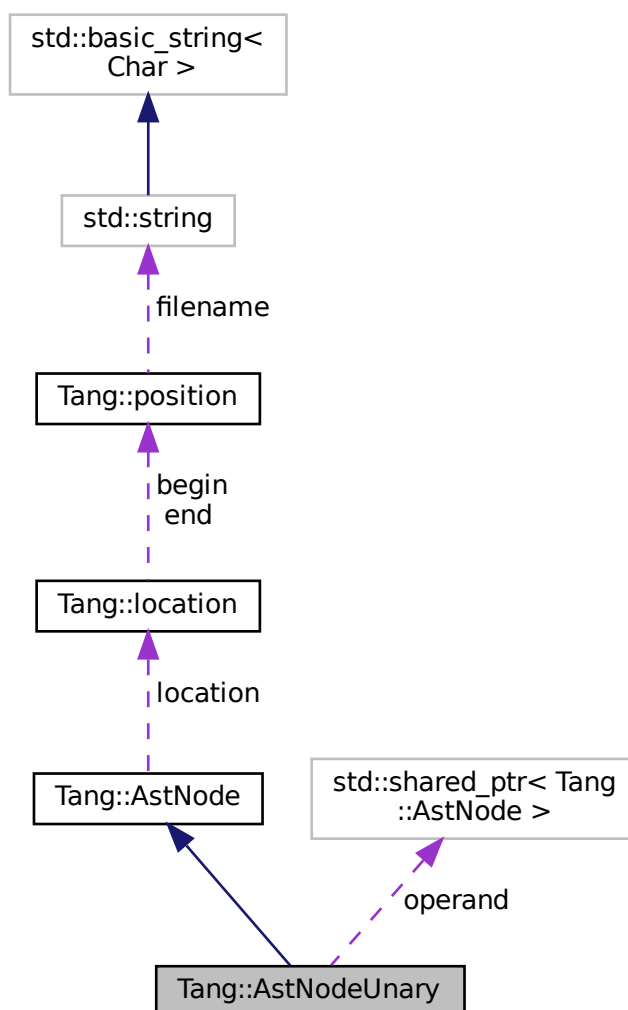
An [AstNode](#) that represents a unary negation.

```
#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.

Private Attributes

- [Operator](#) op
The operation which will be applied to the operand.
- shared_ptr< [AstNode](#) > operand
The operand to which the operation will be applied.
- [Tang::location](#) location
The location associated with this node.

5.25.1 Detailed Description

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

5.25.2 Member Enumeration Documentation

5.25.2.1 Operator

enum [Tang::AstNodeUnary::Operator](#)

The type of operation.

Enumerator

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

5.25.2.2 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.

5.25.3 Constructor & Destructor Documentation

5.25.3.1 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.

5.25.4 Member Function Documentation

5.25.4.1 compile()

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

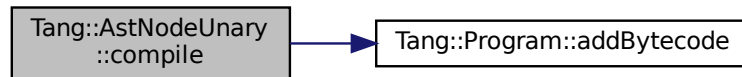
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.25.4.2 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](#).

5.25.4.3 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:

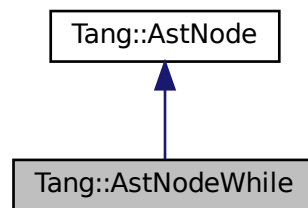
- include/[astNodeUnary.hpp](#)
- src/[astNodeUnary.cpp](#)

5.26 Tang::AstNodeWhile Class Reference

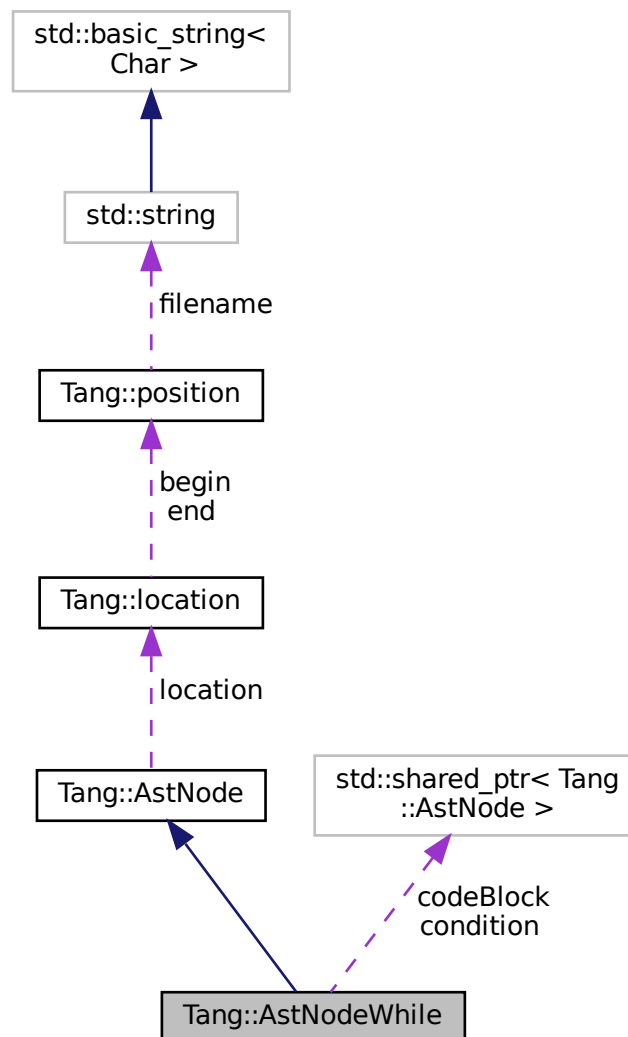
An [AstNode](#) that represents a while statement.

```
#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.

Private Attributes

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

5.26.1 Detailed Description

An `AstNode` that represents a while statement.

5.26.2 Member Enumeration Documentation

5.26.2.1 PreprocessState

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

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

Enumerator

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

5.26.3 Constructor & Destructor Documentation

5.26.3.1 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.

5.26.4 Member Function Documentation

5.26.4.1 compile()

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

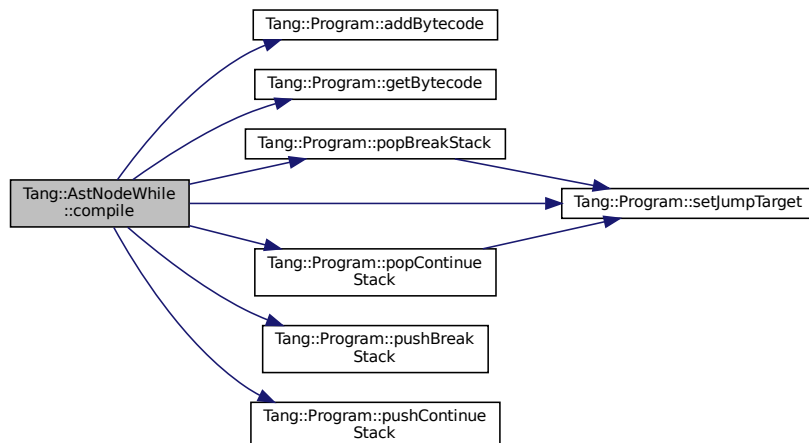
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.26.4.2 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](#).

5.26.4.3 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:

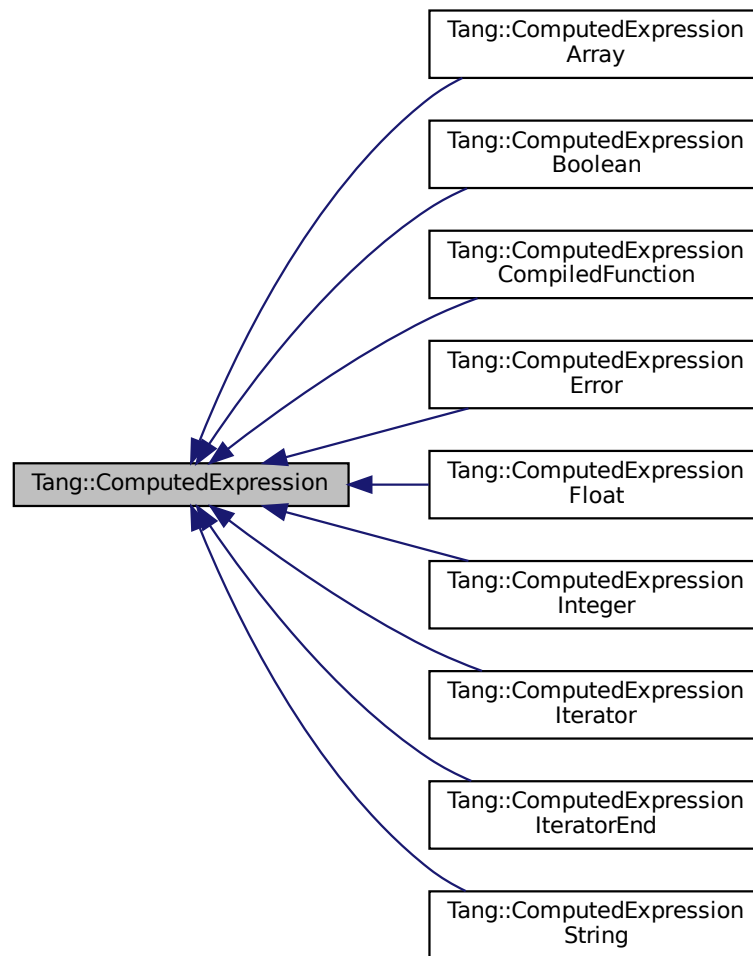
- [include/astNodeWhile.hpp](#)
- [src/astNodeWhile.cpp](#)

5.27 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.
- Compute the result of adding this value and the supplied value.*

 - virtual `GarbageCollected` `__add` (const `GarbageCollected` &rhs) const
- Compute the result of subtracting this value and the supplied value.*

 - virtual `GarbageCollected` `__subtract` (const `GarbageCollected` &rhs) const
- Compute the result of multiplying this value and the supplied value.*

 - virtual `GarbageCollected` `__multiply` (const `GarbageCollected` &rhs) const
- Compute the result of dividing this value and the supplied value.*

 - virtual `GarbageCollected` `__divide` (const `GarbageCollected` &rhs) const
- Compute the result of moduloing this value and the supplied value.*

 - virtual `GarbageCollected` `__modulo` (const `GarbageCollected` &rhs) const
- Compute the result of negating this value.*

 - virtual `GarbageCollected` `__negative` () const
- Compute the logical not of this value.*

 - virtual `GarbageCollected` `__not` () const
- Compute the "less than" comparison.*

 - virtual `GarbageCollected` `__lessThan` (const `GarbageCollected` &rhs) const
- Perform an equality test.*

 - virtual `GarbageCollected` `__equal` (const `GarbageCollected` &rhs) const
- Perform an index operation.*

 - virtual `GarbageCollected` `__index` (const `GarbageCollected` &index) const
- Perform a slice operation.*

 - virtual `GarbageCollected` `__slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const
- Get an iterator for the expression.*

 - virtual `GarbageCollected` `__getIterator` (const `GarbageCollected` &collection) const
- Get the next iterative value.*

 - virtual `GarbageCollected` `__iteratorNext` (size_t index=0) const
- Perform a type cast to integer.*

 - virtual `GarbageCollected` `__integer` () const
- Perform a type cast to float.*

 - virtual `GarbageCollected` `__float` () const
- Perform a type cast to boolean.*

 - virtual `GarbageCollected` `__boolean` () const
- Perform a type cast to string.*

 - virtual `GarbageCollected` `__string` () const

5.27.1 Detailed Description

Represents the result of a computation that has been executed.

By default, it will represent a NULL value.

5.27.2 Member Function Documentation

5.27.2.1 __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](#).

5.27.2.2 __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](#).

5.27.2.3 __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](#).

5.27.2.4 __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](#).

5.27.2.5 __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](#).

5.27.2.6 __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](#).

5.27.2.7 __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](#).

5.27.2.8 __getIterator()

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

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

5.27.2.9 __index()

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.27.2.10 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.27.2.11 __iteratorNext()

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual]
```

Get the next iterative value.

Parameters

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

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

5.27.2.12 __lessThan()

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

Compute the "less than" comparison.

Parameters

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

Returns

The result of the the operation.

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

5.27.2.13 `__modulo()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.27.2.14 `__multiply()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.27.2.15 `__negative()`

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.27.2.16 `__not()`

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.27.2.17 `__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](#).

5.27.2.18 __string()

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

Perform a type cast to string.

Returns

The result of the the operation.

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

5.27.2.19 __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](#).

5.27.2.20 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::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionIterator](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionComplex](#), [Tang::ComputedExpressionBoolean](#), and [Tang::ComputedExpressionArray](#).

5.27.2.21 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](#).

5.27.2.22 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](#).

5.27.2.23 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.

5.27.2.24 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](#).

5.27.2.25 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](#).

5.27.2.26 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](#).

5.27.2.27 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](#).

5.27.2.28 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:

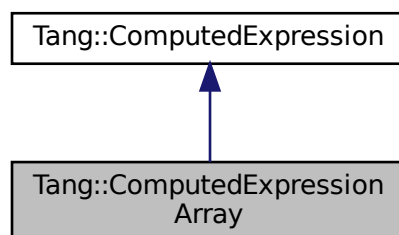
- [include/computedExpression.hpp](#)
- [src/computedExpression.cpp](#)

5.28 Tang::ComputedExpressionArray Class Reference

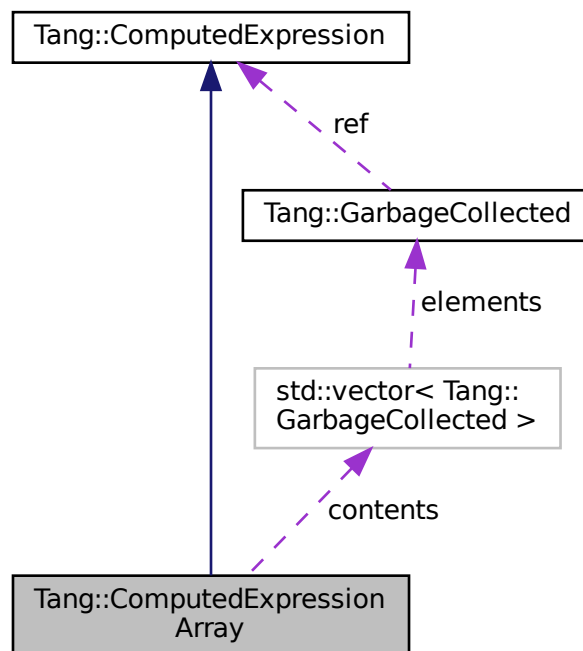
Represents an Array that is the result of a computation.

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

Inheritance diagram for Tang::ComputedExpressionArray:



Collaboration diagram for Tang::ComputedExpressionArray:



Public Member Functions

- **ComputedExpressionArray** (std::vector< **Tang::GarbageCollected** > contents)
Construct an Array result.
- virtual std::string **dump** () const override
Output the contents of the **ComputedExpression** as a string.
- virtual bool **isCopyNeeded** () const override
Determine whether or not a copy is needed.
- **GarbageCollected makeCopy** () const override
Make a copy of the **ComputedExpression** (recursively, if appropriate).
- virtual **GarbageCollected __index** (const **GarbageCollected** &index) const override
Perform an index operation.
- virtual **GarbageCollected __slice** (const **GarbageCollected** &begin, const **GarbageCollected** &end, const **GarbageCollected** &skip) const override
Perform a slice operation.
- virtual **GarbageCollected __getIterator** (const **GarbageCollected** &collection) const override
Get an iterator for the expression.
- virtual **GarbageCollected __iteratorNext** (size_t index) const override
Get the next iterative value.
- virtual **GarbageCollected __assign_index** (const **GarbageCollected** &index, const **GarbageCollected** &value) override
Perform an index assignment to the supplied value.
- virtual **GarbageCollected __string** () const override

- 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.

Private Attributes

- std::vector< [Tang::GarbageCollected](#) > `contents`
- The array contents.*

5.28.1 Detailed Description

Represents an Array that is the result of a computation.

5.28.2 Constructor & Destructor Documentation

5.28.2.1 ComputedExpressionArray()

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

Construct an Array result.

Parameters

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

Returns

The result of the operation.

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

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

Perform an index assignment to the supplied value.

Parameters

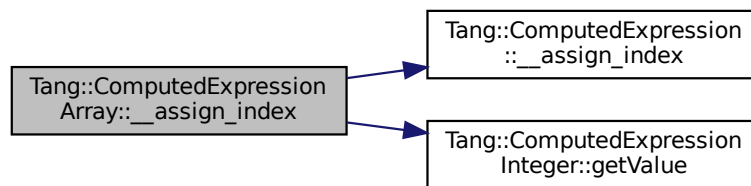
<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

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

Here is the call graph for this function:



5.28.3.4 __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](#).

5.28.3.5 __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 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](#).

5.28.3.7 `__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](#).

5.28.3.8 `__getIterator()`

```
GarbageCollected ComputedExpressionArray::__getIterator (
    const GarbageCollected & collection ) const [override], [virtual]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

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

5.28.3.9 __index()

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

Perform an index operation.

Parameters

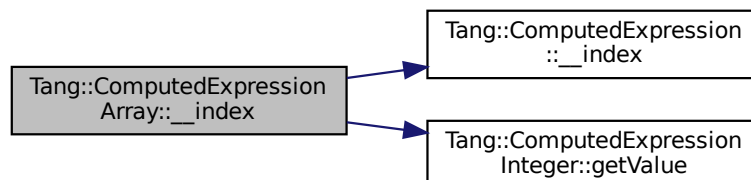
<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:



5.28.3.10 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.28.3.11 `__iteratorNext()`

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

Get the next iterative value.

Parameters

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

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

5.28.3.12 `__lessThan()`

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

Compute the "less than" comparison.

Parameters

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

Returns

The result of the the operation.

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

5.28.3.13 `__modulo()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.28.3.14 __multiply()

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

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

Parameters

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

Returns

The result of the operation.

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

5.28.3.15 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.28.3.16 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.28.3.17 `__slice()`

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

Perform a slice operation.

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

Parameters

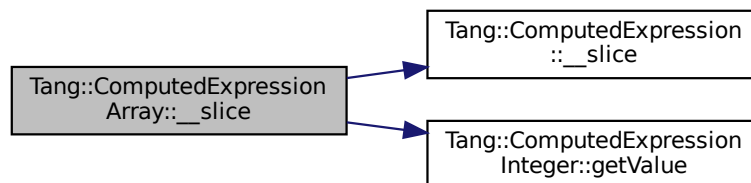
<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

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

Here is the call graph for this function:



5.28.3.18 `__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:

**5.28.3.19 __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](#).

5.28.3.20 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](#).

5.28.3.21 `is_equal()` [1/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.28.3.22 `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](#).

5.28.3.23 `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.

5.28.3.24 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](#).

5.28.3.25 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](#).

5.28.3.26 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](#).

5.28.3.27 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](#).

5.28.3.28 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:

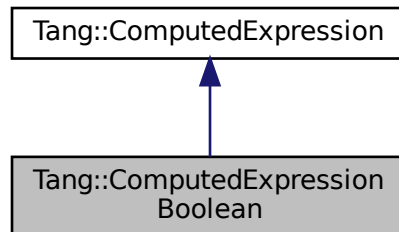
- [include/computedExpressionArray.hpp](#)
- [src/computedExpressionArray.cpp](#)

5.29 Tang::ComputedExpressionBoolean Class Reference

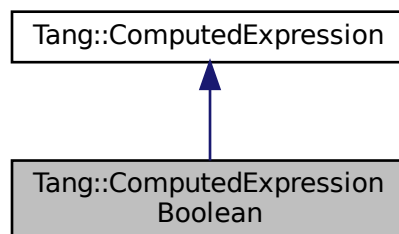
Represents an Boolean that is the result of a computation.

```
#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 __getIterator` (const `GarbageCollected` &collection) const
Get an iterator for the expression.
- virtual `GarbageCollected __iteratorNext` (size_t index=0) const
Get the next iterative value.
- virtual `GarbageCollected __string` () const
Perform a type cast to string.

Private Attributes

- bool `val`

The boolean value.

5.29.1 Detailed Description

Represents an Boolean that is the result of a computation.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 ComputedExpressionBoolean()

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

Construct an Boolean result.

Parameters

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

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

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

Returns

The result of the operation.

Reimplemented in `Tang::ComputedExpressionString`, `Tang::ComputedExpressionInteger`, `Tang::ComputedExpressionFloat`, and `Tang::ComputedExpressionError`.

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

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

Perform an equality test.

Parameters

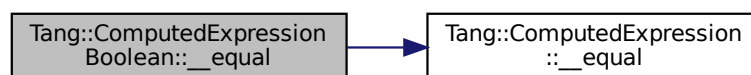
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:

5.29.3.7 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.29.3.8 `__getIterator()`

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

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

5.29.3.9 `__index()`

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.29.3.10 `__integer()`

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.29.3.11 `__iteratorNext()`

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

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

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

5.29.3.12 __lessThan()

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

Compute the "less than" comparison.

Parameters

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

Returns

The result of the the operation.

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

5.29.3.13 __modulo()

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

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

Parameters

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

Returns

The result of the operation.

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

5.29.3.14 __multiply()

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

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

Parameters

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

Returns

The result of the operation.

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

5.29.3.15 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.29.3.16 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.29.3.17 __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](#).

5.29.3.18 `__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::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.29.3.19 `__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](#).

5.29.3.20 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](#).

5.29.3.21 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](#).

5.29.3.22 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](#).

5.29.3.23 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.

5.29.3.24 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](#).

5.29.3.25 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](#).

5.29.3.26 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](#).

5.29.3.27 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](#).

5.29.3.28 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:

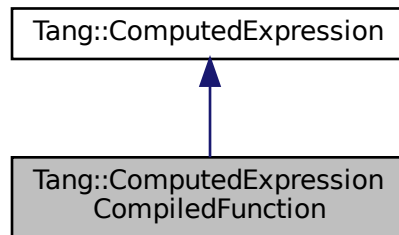
- include/computedExpressionBoolean.hpp
- src/computedExpressionBoolean.cpp

5.30 Tang::ComputedExpressionCompiledFunction Class Reference

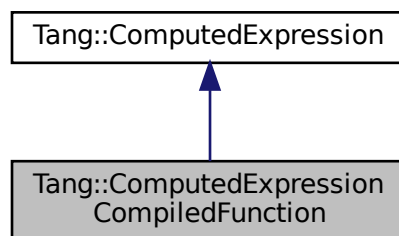
Represents a Compiled Function declared in the script.

```
#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 __getIterator \(const GarbageCollected &collection\)](#) const
Get an iterator for the expression.
- virtual [GarbageCollected __iteratorNext \(size_t index=0\)](#) const
Get the next iterative value.
- virtual [GarbageCollected __integer \(\)](#) const
Perform a type cast to integer.
- virtual [GarbageCollected __float \(\)](#) const
Perform a type cast to float.
- virtual [GarbageCollected __boolean \(\)](#) const
Perform a type cast to boolean.
- virtual [GarbageCollected __string \(\)](#) const
Perform a type cast to string.

Private Attributes

- `uint32_t argc`
The count of arguments that this function expects.
- `Tang::integer_t pc`
The bytecode address of the start of the function.

5.30.1 Detailed Description

Represents a Compiled Function declared in the script.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 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.

5.30.3 Member Function Documentation

5.30.3.1 __add()

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

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

Parameters

<i>rhs</i>	The <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](#).

5.30.3.2 __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 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](#).

5.30.3.5 __divide()

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

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

Parameters

<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](#).

5.30.3.6 __equal()

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

Perform an equality test.

Parameters

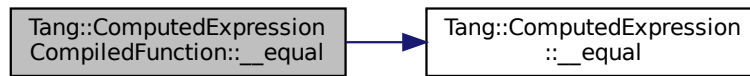
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.30.3.7 __float()

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.30.3.8 __getIterator()

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

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

5.30.3.9 __index()

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.30.3.10 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.30.3.11 __iteratorNext()

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

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

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

5.30.3.12 __lessThan()

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

Compute the "less than" comparison.

Parameters

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

Returns

The result of the the operation.

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

5.30.3.13 `__modulo()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.30.3.14 `__multiply()`

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

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

Parameters

<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](#).

5.30.3.15 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.30.3.16 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.30.3.17 __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](#).

5.30.3.18 __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::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.30.3.19 __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](#).

5.30.3.20 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](#).

5.30.3.21 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](#).

5.30.3.22 is_equal() [2/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.30.3.23 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.

5.30.3.24 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](#).

5.30.3.25 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](#).

5.30.3.26 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](#).

5.30.3.27 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](#).

5.30.3.28 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:

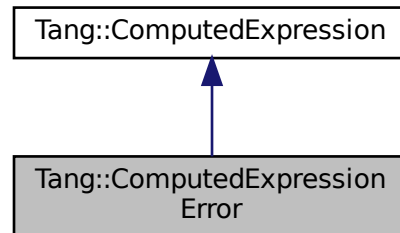
- [include/computedExpressionCompiledFunction.hpp](#)
- [src/computedExpressionCompiledFunction.cpp](#)

5.31 Tang::ComputedExpressionError Class Reference

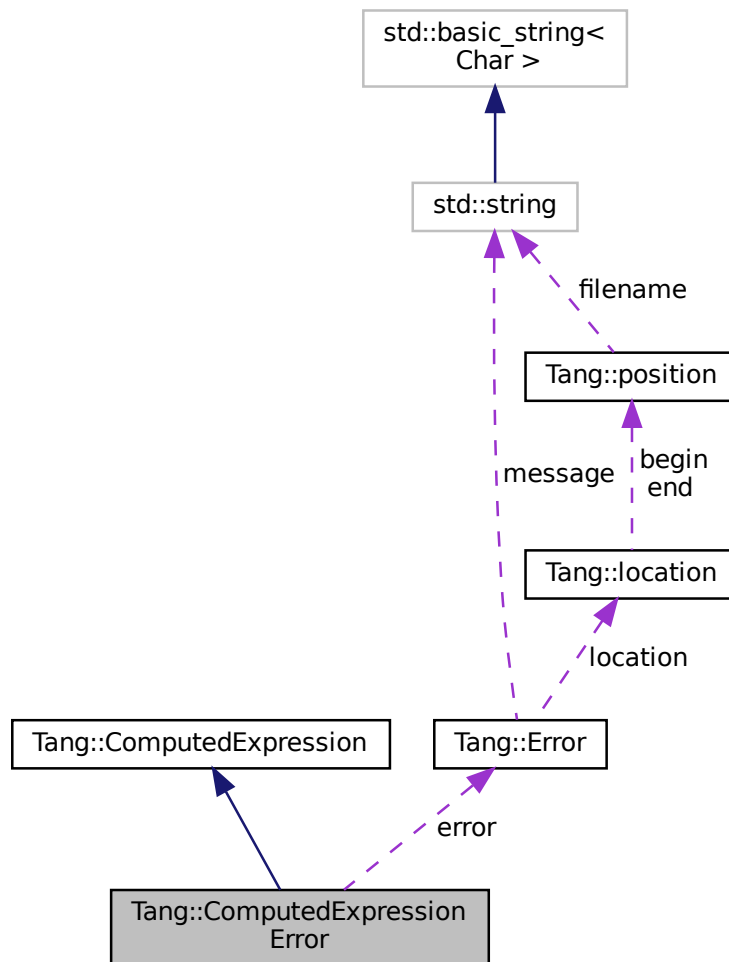
Represents a Runtime [Error](#).

```
#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.*

 - virtual [GarbageCollected __getIterator](#) (const [GarbageCollected](#) &collection) const
- Get an iterator for the expression.*

 - virtual [GarbageCollected __iteratorNext](#) (size_t index=0) const
- Get the next iterative value.*

Private Attributes

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

5.31.1 Detailed Description

Represents a Runtime [Error](#).

5.31.2 Constructor & Destructor Documentation

5.31.2.1 ComputedExpressionError()

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

Construct a Runtime [Error](#).

Parameters

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

5.31.3 Member Function Documentation

5.31.3.1 __add()

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

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

Parameters

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

Returns

The result of the operation.

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

5.31.3.2 __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 ComputedExpressionError::__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 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](#).

5.31.3.6 __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](#).

5.31.3.7 __float()

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.31.3.8 __getIterator()

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

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

5.31.3.9 __index()

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.31.3.10 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.31.3.11 __iteratorNext()

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

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

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

5.31.3.12 __lessThan()

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

Compute the "less than" comparison.

Parameters

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

Returns

The result of the the operation.

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

5.31.3.13 __modulo()

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

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

Parameters

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

Returns

The result of the operation.

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

5.31.3.14 __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](#).

5.31.3.15 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.31.3.16 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.31.3.17 __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](#).

5.31.3.18 `__string()`

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

Perform a type cast to string.

Returns

The result of the the operation.

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

5.31.3.19 `__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](#).

5.31.3.20 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](#).

5.31.3.21 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](#).

5.31.3.22 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](#).

5.31.3.23 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.

5.31.3.24 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](#).

5.31.3.25 is_equal() [5/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.31.3.26 is_equal() [6/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.31.3.27 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](#).

5.31.3.28 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:

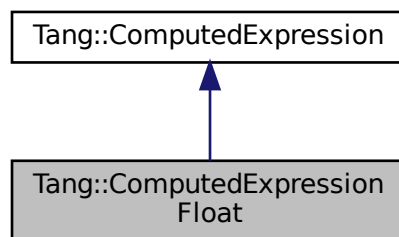
- [include/computedExpressionError.hpp](#)
- [src/computedExpressionError.cpp](#)

5.32 Tang::ComputedExpressionFloat Class Reference

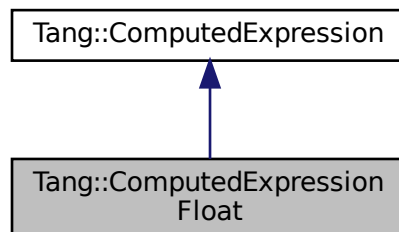
Represents a Float that is the result of a computation.

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

Inheritance diagram for Tang::ComputedExpressionFloat:



Collaboration diagram for Tang::ComputedExpressionFloat:



Public Member Functions

- [ComputedExpressionFloat](#) ([Tang::float_t](#) val)
Construct a Float result.
- virtual std::string [dump](#) () const override
Output the contents of the [ComputedExpression](#) as a string.
- [GarbageCollected](#) [makeCopy](#) () const override
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual bool [is_equal](#) (const [Tang::integer_t](#) &val) const override
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Tang::float_t](#) &val) const override
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const bool &val) const override
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected](#) [__add](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of adding this value and the supplied value.
- virtual [GarbageCollected](#) [__subtract](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of subtracting this value and the supplied value.
- virtual [GarbageCollected](#) [__multiply](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of multiplying this value and the supplied value.
- virtual [GarbageCollected](#) [__divide](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of dividing this value and the supplied value.
- virtual [GarbageCollected](#) [__negative](#) () const override
Compute the result of negating this value.
- virtual [GarbageCollected](#) [__not](#) () const override
Compute the logical not of this value.
- virtual [GarbageCollected](#) [__lessThan](#) (const [GarbageCollected](#) &rhs) const override
Compute the "less than" comparison.
- virtual [GarbageCollected](#) [__equal](#) (const [GarbageCollected](#) &rhs) const override
Perform an equality test.
- virtual [GarbageCollected](#) [__integer](#) () const override
Perform a type cast to integer.
- virtual [GarbageCollected](#) [__float](#) () const override
Perform a type cast to float.
- virtual [GarbageCollected](#) [__boolean](#) () const override
Perform a type cast to boolean.
- virtual [GarbageCollected](#) [__string](#) () const override
Perform a type cast to string.
- [Tang::float_t](#) [getValue](#) () const
Helper function to get the value associated with this expression.
- virtual std::string [__asCode](#) () const
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- virtual bool [isCopyNeeded](#) () const
Determine whether or not a copy is needed.
- virtual bool [is_equal](#) (const string &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Error](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected](#) [__assign_index](#) (const [GarbageCollected](#) &index, const [GarbageCollected](#) &value)

- virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const
Perform an index assignment to the supplied value.
- virtual `GarbageCollected __index` (const `GarbageCollected` &index) const
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const
Perform an index operation.
- virtual `GarbageCollected __getIterator` (const `GarbageCollected` &collection) const
Perform a slice operation.
- virtual `GarbageCollected __iteratorNext` (size_t index=0) const
Get an iterator for the expression.
- virtual `GarbageCollected __iteratorNext` (size_t index=0) const
Get the next iterative value.

Private Attributes

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

5.32.1 Detailed Description

Represents a Float that is the result of a computation.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 ComputedExpressionFloat()

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

Construct a Float result.

Parameters

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

5.32.3 Member Function Documentation

5.32.3.1 __add()

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

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

Parameters

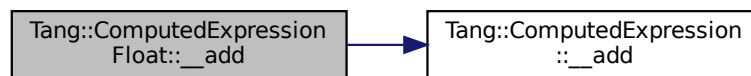
<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:



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

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

Parameters

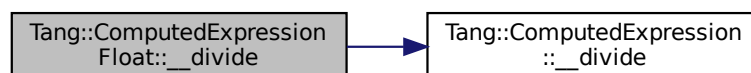
<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.32.3.6 `__equal()`

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

Perform an equality test.

Parameters

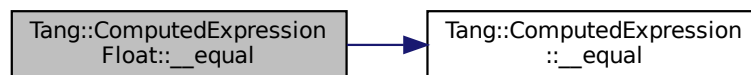
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:

5.32.3.7 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.32.3.8 `__getIterator()`

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

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

5.32.3.9 `__index()`

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.32.3.10 `__integer()`

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.32.3.11 `__iteratorNext()`

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

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

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

5.32.3.12 __lessThan()

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

Compute the "less than" comparison.

Parameters

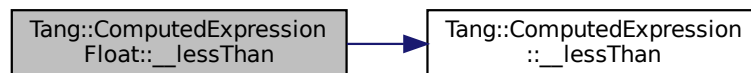
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.32.3.13 __modulo()

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

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

Parameters

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

Returns

The result of the operation.

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

5.32.3.14 __multiply()

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

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

Parameters

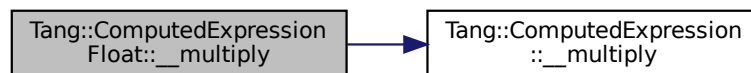
<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:

**5.32.3.15 __negative()**

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.32.3.16 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.32.3.17 __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](#).

5.32.3.18 __string()

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

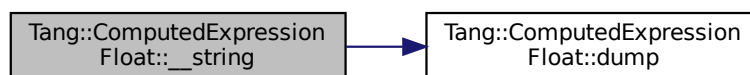
Perform a type cast to string.

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.32.3.19 __subtract()**

```

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

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

Parameters

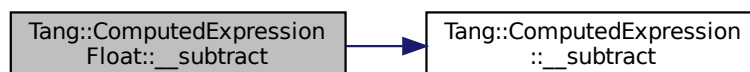
<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.32.3.20 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](#).

5.32.3.21 getValue()

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

Helper function to get the value associated with this expression.

Returns

The value associated with this expression.

5.32.3.22 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](#).

5.32.3.23 is_equal() [2/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.32.3.24 is_equal() [3/6]

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

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

Parameters

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

Returns

True if equal, false if not.

5.32.3.25 is_equal() [4/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.32.3.26 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](#).

5.32.3.27 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](#).

5.32.3.28 isCopyNeeded()

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

Determine whether or not a copy is needed.

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

Returns

Whether or not a copy is needed.

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

5.32.3.29 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:

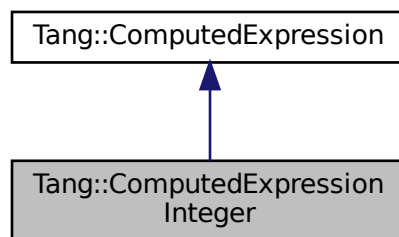
- [include/computedExpressionFloat.hpp](#)
- [src/computedExpressionFloat.cpp](#)

5.33 Tang::ComputedExpressionInteger Class Reference

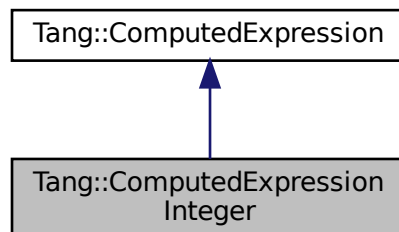
Represents an Integer that is the result of a computation.

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

Inheritance diagram for Tang::ComputedExpressionInteger:



Collaboration diagram for Tang::ComputedExpressionInteger:



Public Member Functions

- [ComputedExpressionInteger](#) ([Tang::integer_t val](#))
Construct an Integer result.
- virtual std::string [dump](#) () const override
Output the contents of the [ComputedExpression](#) as a string.
- [GarbageCollected](#) [makeCopy](#) () const override
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual bool [is_equal](#) (const [Tang::integer_t](#) &[val](#)) const override
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Tang::float_t](#) &[val](#)) const override
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const bool &[val](#)) const override
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected](#) [__add](#) (const [GarbageCollected](#) &[rhs](#)) const override
Compute the result of adding this value and the supplied value.
- virtual [GarbageCollected](#) [__subtract](#) (const [GarbageCollected](#) &[rhs](#)) const override
Compute the result of subtracting this value and the supplied value.
- virtual [GarbageCollected](#) [__multiply](#) (const [GarbageCollected](#) &[rhs](#)) const override
Compute the result of multiplying this value and the supplied value.
- virtual [GarbageCollected](#) [__divide](#) (const [GarbageCollected](#) &[rhs](#)) const override
Compute the result of dividing this value and the supplied value.
- virtual [GarbageCollected](#) [__modulo](#) (const [GarbageCollected](#) &[rhs](#)) const override
Compute the result of moduloing this value and the supplied value.
- virtual [GarbageCollected](#) [__negative](#) () const override
Compute the result of negating this value.
- virtual [GarbageCollected](#) [__not](#) () const override
Compute the logical not of this value.
- virtual [GarbageCollected](#) [__lessThan](#) (const [GarbageCollected](#) &[rhs](#)) const override
Compute the "less than" comparison.
- virtual [GarbageCollected](#) [__equal](#) (const [GarbageCollected](#) &[rhs](#)) const override
Perform an equality test.
- virtual [GarbageCollected](#) [__integer](#) () const override
Perform a type cast to integer.
- virtual [GarbageCollected](#) [__float](#) () const override
Perform a type cast to float.
- virtual [GarbageCollected](#) [__boolean](#) () const override
Perform a type cast to boolean.
- virtual [GarbageCollected](#) [__string](#) () const override
Perform a type cast to string.
- [Tang::integer_t](#) [getValue](#) () const
Helper function to get the value associated with this expression.
- virtual std::string [__asCode](#) () const
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- virtual bool [isCopyNeeded](#) () const
Determine whether or not a copy is needed.
- virtual bool [is_equal](#) (const string &[val](#)) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Error](#) &[val](#)) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const std::nullptr_t &[val](#)) const

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

- virtual `GarbageCollected __assign_index` (const `GarbageCollected` &index, const `GarbageCollected` &value)

Perform an index assignment to the supplied value.

- virtual `GarbageCollected __index` (const `GarbageCollected` &index) const

Perform an index operation.

- virtual `GarbageCollected __slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const

Perform a slice operation.

- virtual `GarbageCollected __getIterator` (const `GarbageCollected` &collection) const

Get an iterator for the expression.

- virtual `GarbageCollected __iteratorNext` (size_t index=0) const

Get the next iterative value.

Private Attributes

- `Tang::integer_t val`

The integer value.

5.33.1 Detailed Description

Represents an Integer that is the result of a computation.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 ComputedExpressionInteger()

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

Construct an Integer result.

Parameters

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

5.33.3 Member Function Documentation

5.33.3.1 __add()

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

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

Parameters

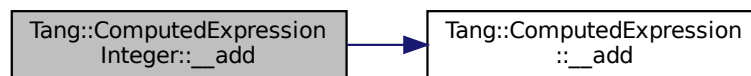
<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:

**5.33.3.2 __asCode()**

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

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

Returns

A code-string representation of the computed expression.

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

5.33.3.3 __assign_index()

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.33.3.4 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.33.3.5 __divide()

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

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

Parameters

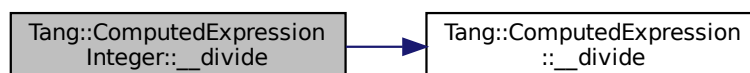
<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.33.3.6 `__equal()`

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

Perform an equality test.

Parameters

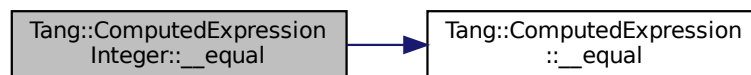
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.33.3.7 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.33.3.8 `__getIterator()`

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

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

5.33.3.9 `__index()`

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.33.3.10 `__integer()`

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.33.3.11 `__iteratorNext()`

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

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

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

5.33.3.12 `__lessThan()`

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

Compute the "less than" comparison.

Parameters

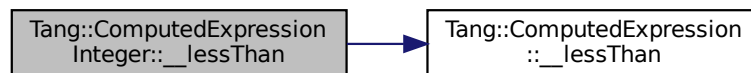
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.33.3.13 `__modulo()`

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

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

Parameters

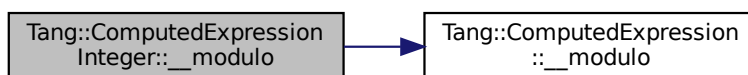
<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:

**5.33.3.14 __multiply()**

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

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

Parameters

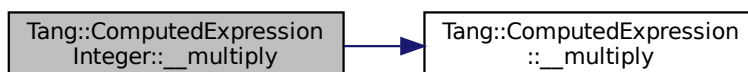
<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.33.3.15 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.33.3.16 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.33.3.17 __slice()

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

Perform a slice operation.

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

Parameters

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

Returns

The result of the operation.

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

5.33.3.18 __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:



5.33.3.19 __subtract()

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

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

Parameters

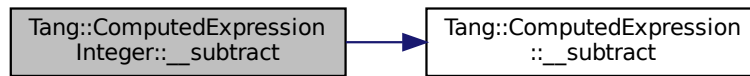
<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.33.3.20 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](#).

5.33.3.21 getValue()

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

Helper function to get the value associated with this expression.

Returns

The value associated with this expression.

5.33.3.22 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](#).

5.33.3.23 is_equal() [2/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.33.3.24 is_equal() [3/6]

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

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

Parameters

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

Returns

True if equal, false if not.

5.33.3.25 is_equal() [4/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.33.3.26 is_equal() [5/6]

```
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](#).

5.33.3.27 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](#).

5.33.3.28 isCopyNeeded()

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

Determine whether or not a copy is needed.

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

Returns

Whether or not a copy is needed.

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

5.33.3.29 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:

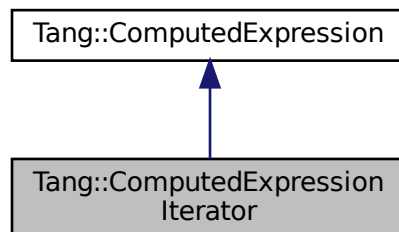
- [include/computedExpressionInteger.hpp](#)
- [src/computedExpressionInteger.cpp](#)

5.34 Tang::ComputedExpressionIterator Class Reference

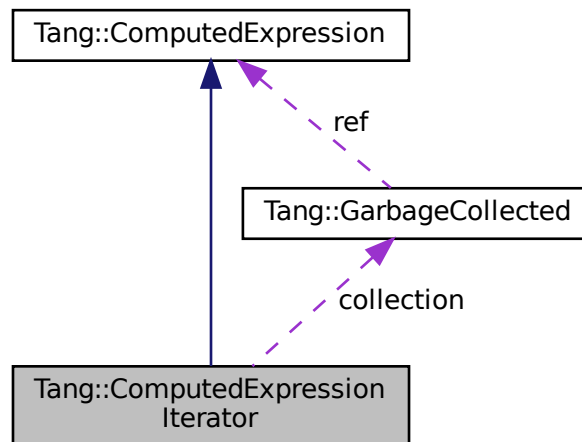
Represents an Iterator that is the result of a computation.

```
#include <computedExpressionIterator.hpp>
```

Inheritance diagram for Tang::ComputedExpressionIterator:



Collaboration diagram for Tang::ComputedExpressionIterator:



Public Member Functions

- [ComputedExpressionIterator](#) ([Tang::GarbageCollected](#) collection)
Construct an Iterator result.
- virtual std::string [dump](#) () const override
Output the contents of the [ComputedExpression](#) as a string.
- virtual [GarbageCollected](#) [__iteratorNext](#) (size_t index) const override
Get the next iterative value.
- virtual std::string [__asCode](#) () const
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- virtual bool [isCopyNeeded](#) () const
Determine whether or not a copy is needed.
- virtual [GarbageCollected](#) [makeCopy](#) () const
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual bool [is_equal](#) (const [Tang::integer_t](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Tang::float_t](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const bool &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const string &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Error](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected](#) [__assign_index](#) (const [GarbageCollected](#) &index, const [GarbageCollected](#) &value)
Perform an index assignment to the supplied value.

- virtual [GarbageCollected __add](#) (const [GarbageCollected](#) &rhs) const
Compute the result of adding this value and the supplied value.
- virtual [GarbageCollected __subtract](#) (const [GarbageCollected](#) &rhs) const
Compute the result of subtracting this value and the supplied value.
- virtual [GarbageCollected __multiply](#) (const [GarbageCollected](#) &rhs) const
Compute the result of multiplying this value and the supplied value.
- virtual [GarbageCollected __divide](#) (const [GarbageCollected](#) &rhs) const
Compute the result of dividing this value and the supplied value.
- virtual [GarbageCollected __modulo](#) (const [GarbageCollected](#) &rhs) const
Compute the result of moduloing this value and the supplied value.
- virtual [GarbageCollected __negative](#) () const
Compute the result of negating this value.
- virtual [GarbageCollected __not](#) () const
Compute the logical not of this value.
- virtual [GarbageCollected __lessThan](#) (const [GarbageCollected](#) &rhs) const
Compute the "less than" comparison.
- virtual [GarbageCollected __equal](#) (const [GarbageCollected](#) &rhs) const
Perform an equality test.
- virtual [GarbageCollected __index](#) (const [GarbageCollected](#) &index) const
Perform an index operation.
- virtual [GarbageCollected __slice](#) (const [GarbageCollected](#) &begin, const [GarbageCollected](#) &end, const [GarbageCollected](#) &skip) const
Perform a slice operation.
- virtual [GarbageCollected __getIterator](#) (const [GarbageCollected](#) &collection) const
Get an iterator for the expression.
- virtual [GarbageCollected __integer](#) () const
Perform a type cast to integer.
- virtual [GarbageCollected __float](#) () const
Perform a type cast to float.
- virtual [GarbageCollected __boolean](#) () const
Perform a type cast to boolean.
- virtual [GarbageCollected __string](#) () const
Perform a type cast to string.

Private Attributes

- [Tang::GarbageCollected collection](#)
The target collection.
- `size_t` [index](#)
The next index.

5.34.1 Detailed Description

Represents an Iterator that is the result of a computation.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 ComputedExpressionIterator()

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

Construct an iterator result.

Parameters

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

5.34.3 Member Function Documentation

5.34.3.1 __add()

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

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

Parameters

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

Returns

The result of the operation.

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

5.34.3.2 __asCode()

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

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

Returns

A code-string representation of the computed expression.

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

5.34.3.3 __assign_index()

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.34.3.4 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.34.3.5 __divide()

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

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

Parameters

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

Returns

The result of the operation.

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

5.34.3.6 `__equal()`

```
GarbageCollected 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](#).

5.34.3.7 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.34.3.8 `__getIterator()`

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

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

5.34.3.9 __index()

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.34.3.10 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.34.3.11 __iteratorNext()

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

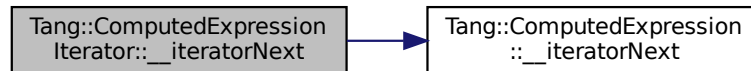
Get the next iterative value.

Parameters

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

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

Here is the call graph for this function:



5.34.3.12 __lessThan()

```

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

Compute the "less than" comparison.

Parameters

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

Returns

The result of the the operation.

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

5.34.3.13 __modulo()

```

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

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

Parameters

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

Returns

The result of the operation.

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

5.34.3.14 __multiply()

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

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

Parameters

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

Returns

The result of the operation.

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

5.34.3.15 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.34.3.16 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.34.3.17 `__slice()`

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

Perform a slice operation.

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

Parameters

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

Returns

The result of the operation.

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

5.34.3.18 `__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::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), and [Tang::ComputedExpressionArray](#).

5.34.3.19 `__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](#).

5.34.3.20 dump()

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

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

Returns

A string representation of the computed expression.

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

5.34.3.21 is_equal() [1/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.34.3.22 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](#).

5.34.3.23 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.

5.34.3.24 is_equal() [4/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.34.3.25 is_equal() [5/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.34.3.26 is_equal() [6/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.34.3.27 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](#).

5.34.3.28 makeCopy()

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

Make a copy of the `ComputedExpression` (recursively, if appropriate).

Returns

A `Tang::GarbageCollected` value for the new `ComputedExpression`.

Reimplemented in `Tang::ComputedExpressionString`, `Tang::ComputedExpressionInteger`, `Tang::ComputedExpressionFloat`, `Tang::ComputedExpressionError`, `Tang::ComputedExpressionCompiledFunction`, `Tang::ComputedExpressionBoolean`, and `Tang::ComputedExpressionArray`.

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

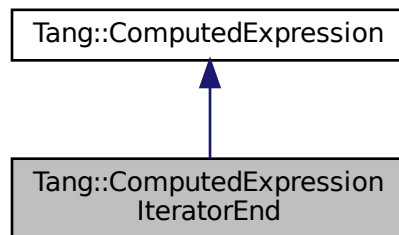
- include/computedExpressionIterator.hpp
- src/computedExpressionIterator.cpp

5.35 Tang::ComputedExpressionIteratorEnd Class Reference

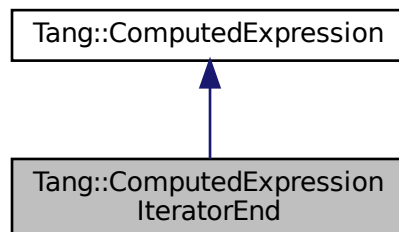
Represents that a collection has no more values through which to iterate.

```
#include <computedExpressionIteratorEnd.hpp>
```

Inheritance diagram for Tang::ComputedExpressionIteratorEnd:



Collaboration diagram for Tang::ComputedExpressionIteratorEnd:



Public Member Functions

- [ComputedExpressionIteratorEnd](#) ()
Construct an IteratorEnd result.
- virtual std::string [dump](#) () const override
Output the contents of the [ComputedExpression](#) as a string.
- virtual [GarbageCollected __string](#) () const override
Perform a type cast to string.
- virtual std::string [__asCode](#) () const
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- virtual bool [isCopyNeeded](#) () const
Determine whether or not a copy is needed.
- virtual [GarbageCollected makeCopy](#) () const
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual bool [is_equal](#) (const [Tang::integer_t](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Tang::float_t](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const bool &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const string &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const [Error](#) &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const std::nullptr_t &val) const
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected __assign_index](#) (const [GarbageCollected](#) &index, const [GarbageCollected](#) &value)
Perform an index assignment to the supplied value.
- virtual [GarbageCollected __add](#) (const [GarbageCollected](#) &rhs) const
Compute the result of adding this value and the supplied value.
- virtual [GarbageCollected __subtract](#) (const [GarbageCollected](#) &rhs) const
Compute the result of subtracting this value and the supplied value.
- virtual [GarbageCollected __multiply](#) (const [GarbageCollected](#) &rhs) const
Compute the result of multiplying this value and the supplied value.
- virtual [GarbageCollected __divide](#) (const [GarbageCollected](#) &rhs) const
Compute the result of dividing this value and the supplied value.
- virtual [GarbageCollected __modulo](#) (const [GarbageCollected](#) &rhs) const
Compute the result of moduloing this value and the supplied value.
- virtual [GarbageCollected __negative](#) () const
Compute the result of negating this value.
- virtual [GarbageCollected __not](#) () const
Compute the logical not of this value.
- virtual [GarbageCollected __lessThan](#) (const [GarbageCollected](#) &rhs) const
Compute the "less than" comparison.
- virtual [GarbageCollected __equal](#) (const [GarbageCollected](#) &rhs) const
Perform an equality test.
- virtual [GarbageCollected __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](#) [__iterator](#) (const [GarbageCollected](#) &collection) const
Get an iterator for the expression.
- virtual [GarbageCollected](#) [__iteratorNext](#) (size_t index=0) const
Get the next iterative value.
- virtual [GarbageCollected](#) [__integer](#) () const
Perform a type cast to integer.
- virtual [GarbageCollected](#) [__float](#) () const
Perform a type cast to float.
- virtual [GarbageCollected](#) [__boolean](#) () const
Perform a type cast to boolean.

5.35.1 Detailed Description

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

5.35.2 Member Function Documentation

5.35.2.1 [__add\(\)](#)

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

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

Parameters

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

Returns

The result of the operation.

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

5.35.2.2 [__asCode\(\)](#)

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

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

Returns

A code-string representation of the computed expression.

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

5.35.2.3 `__assign_index()`

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.35.2.4 `__boolean()`

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.35.2.5 `__divide()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.35.2.6 __equal()

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

Perform an equality test.

Parameters

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

Returns

The result of the the operation.

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

5.35.2.7 __float()

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.35.2.8 __getIterator()

```
GarbageCollected ComputedExpression::__getIterator (
    const GarbageCollected & collection ) const [virtual], [inherited]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

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

5.35.2.9 __index()

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.35.2.10 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.35.2.11 __iteratorNext()

```
GarbageCollected ComputedExpression::__iteratorNext (
    size_t index = 0 ) const [virtual], [inherited]
```

Get the next iterative value.

Parameters

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

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

5.35.2.12 __lessThan()

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

Compute the "less than" comparison.

Parameters

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

Returns

The result of the the operation.

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

5.35.2.13 __modulo()

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

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

Parameters

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

Returns

The result of the operation.

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

5.35.2.14 __multiply()

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

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

Parameters

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

Returns

The result of the operation.

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

5.35.2.15 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.35.2.16 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.35.2.17 __slice()

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

Perform a slice operation.

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

Parameters

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

Returns

The result of the operation.

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

5.35.2.18 `__string()`

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

Perform a type cast to string.

Returns

The result of the the operation.

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

5.35.2.19 `__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](#).

5.35.2.20 dump()

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

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

Returns

A string representation of the computed expression.

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

5.35.2.21 is_equal() [1/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.35.2.22 is_equal() [2/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.35.2.23 is_equal() [3/6]

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

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

Parameters

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

Returns

True if equal, false if not.

5.35.2.24 is_equal() [4/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.35.2.25 is_equal() [5/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.35.2.26 is_equal() [6/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.35.2.27 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](#).

5.35.2.28 makeCopy()

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

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

Returns

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

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

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

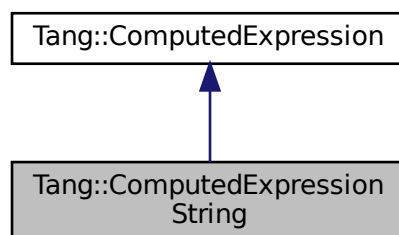
- include/computedExpressionIteratorEnd.hpp
- src/computedExpressionIteratorEnd.cpp

5.36 Tang::ComputedExpressionString Class Reference

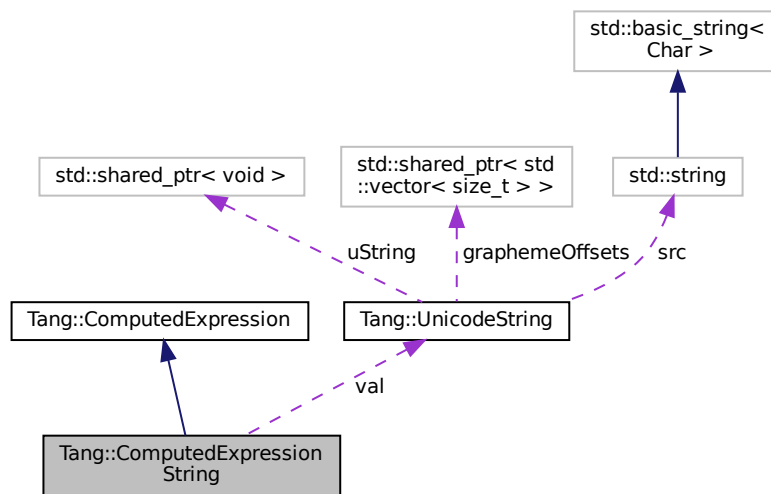
Represents a String that is the result of a computation.

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

Inheritance diagram for Tang::ComputedExpressionString:



Collaboration diagram for Tang::ComputedExpressionString:



Public Member Functions

- [ComputedExpressionString](#) (std::string val)
Construct a String result.
- virtual std::string [dump](#) () const override
Output the contents of the [ComputedExpression](#) as a string.
- virtual std::string [__asCode](#) () const override
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- [GarbageCollected](#) [makeCopy](#) () const override
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual bool [is_equal](#) (const bool &val) const override
Check whether or not the computed expression is equal to another value.
- virtual bool [is_equal](#) (const string &val) const override
Check whether or not the computed expression is equal to another value.
- virtual [GarbageCollected](#) [__index](#) (const [GarbageCollected](#) &index) const override
Perform an index operation.
- virtual [GarbageCollected](#) [__slice](#) (const [GarbageCollected](#) &begin, const [GarbageCollected](#) &end, const [GarbageCollected](#) &skip) const override
Perform a slice operation.
- virtual [GarbageCollected](#) [__getIterator](#) (const [GarbageCollected](#) &collection) const override
Get an iterator for the expression.
- virtual [GarbageCollected](#) [__iteratorNext](#) (size_t index) const override
Get the next iterative value.
- virtual [GarbageCollected](#) [__add](#) (const [GarbageCollected](#) &rhs) const override
Compute the result of adding this value and the supplied value.
- virtual [GarbageCollected](#) [__not](#) () const override
Compute the logical not of this value.
- virtual [GarbageCollected](#) [__lessThan](#) (const [GarbageCollected](#) &rhs) const override
Compute the "less than" comparison.

- virtual [GarbageCollected](#) `__equal` (const [GarbageCollected](#) &rhs) const override
Perform an equality test.
- virtual [GarbageCollected](#) `__boolean` () const override
Perform a type cast to boolean.
- virtual [GarbageCollected](#) `__string` () const override
Perform a type cast to string.
- 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.

Private Attributes

- [UnicodeString](#) val
The string value.

5.36.1 Detailed Description

Represents a String that is the result of a computation.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 ComputedExpressionString()

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

Construct a String result.

Parameters

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

5.36.3 Member Function Documentation

5.36.3.1 `__add()`

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

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

Parameters

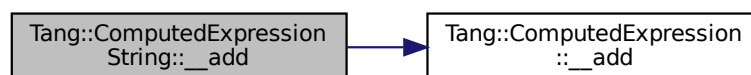
<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:



5.36.3.2 `__asCode()`

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

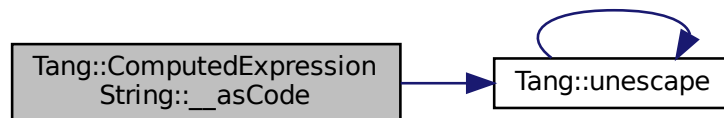
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

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

Here is the call graph for this function:

**5.36.3.3 __assign_index()**

```

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.36.3.4 __boolean()

```

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

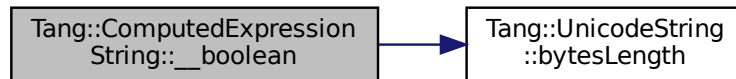
Perform a type cast to boolean.

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.36.3.5 __divide()**

```

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

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

Parameters

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

Returns

The result of the operation.

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

5.36.3.6 __equal()

```

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

Perform an equality test.

Parameters

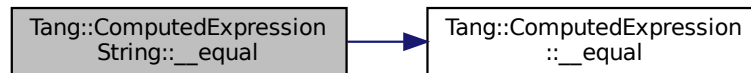
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.36.3.7 __float()**

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.36.3.8 __getlterator()

```
GarbageCollected ComputedExpressionString::__getlterator (
    const GarbageCollected & collection ) const [override], [virtual]
```

Get an iterator for the expression.

Parameters

<i>collection</i>	The GarbageCollected value that will serve as the collection through which to iterate.
-------------------	--

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

5.36.3.9 `__index()`

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

Perform an index operation.

Parameters

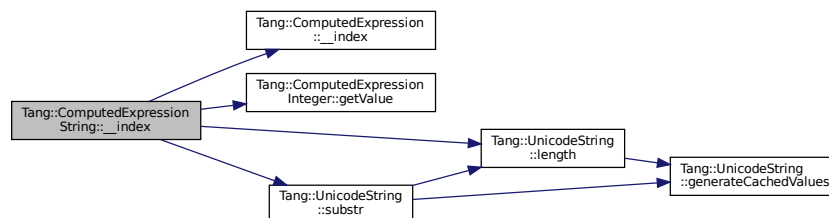
<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:



5.36.3.10 `__integer()`

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.36.3.11 `__iteratorNext()`

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

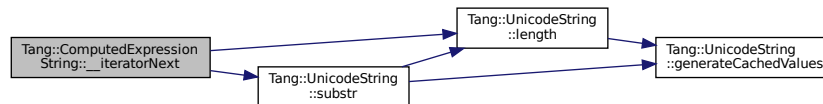
Get the next iterative value.

Parameters

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

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

Here is the call graph for this function:



5.36.3.12 __lessThan()

```

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

Compute the "less than" comparison.

Parameters

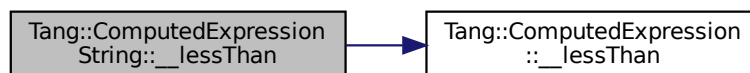
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.36.3.13 __modulo()

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

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

Parameters

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

Returns

The result of the operation.

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

5.36.3.14 __multiply()

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

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

Parameters

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

Returns

The result of the operation.

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

5.36.3.15 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.36.3.16 `__not()`

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

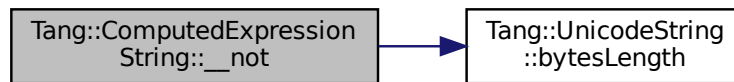
Compute the logical not of this value.

Returns

The result of the operation.

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

Here is the call graph for this function:

**5.36.3.17** `__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

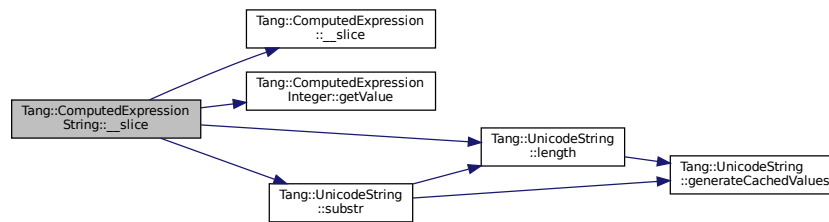
<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

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

Here is the call graph for this function:



5.36.3.18 `__string()`

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

Perform a type cast to string.

Returns

The result of the the operation.

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

5.36.3.19 `__subtract()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.36.3.20 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](#).

5.36.3.21 is_equal() [1/6]

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

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

Parameters

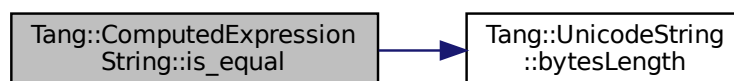
<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

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

Here is the call graph for this function:

**5.36.3.22 is_equal() [2/6]**

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.36.3.23 is_equal() [3/6]

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

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

Parameters

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

Returns

True if equal, false if not.

5.36.3.24 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](#).

5.36.3.25 is_equal() [5/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.36.3.26 is_equal() [6/6]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.36.3.27 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](#).

5.36.3.28 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:

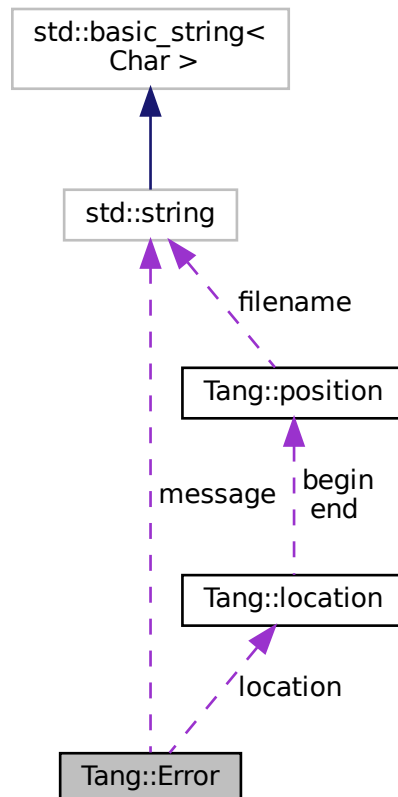
- include/computedExpressionString.hpp
- src/computedExpressionString.cpp

5.37 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.

5.37.1 Detailed Description

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

5.37.2 Constructor & Destructor Documentation

5.37.2.1 [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.
----------------	--------------------------------

5.37.2.2 [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.

5.37.3 Friends And Related Function Documentation

5.37.3.1 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:

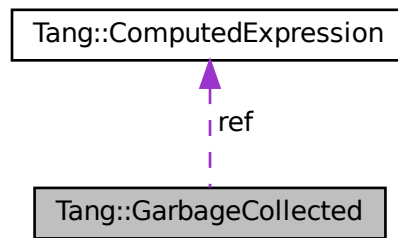
- [include/error.hpp](#)
- [src/error.cpp](#)

5.38 Tang::GarbageCollected Class Reference

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

```
#include <garbageCollected.hpp>
```

Collaboration diagram for Tang::GarbageCollected:



Public Member Functions

- [GarbageCollected](#) (const [GarbageCollected](#) &other)
Copy Constructor.
- [GarbageCollected](#) ([GarbageCollected](#) &&other)
Move Constructor.
- [GarbageCollected](#) & operator= (const [GarbageCollected](#) &other)
Copy Assignment.
- [GarbageCollected](#) & operator= ([GarbageCollected](#) &&other)
Move Assignment.
- ~[GarbageCollected](#) ()
Destructor.
- bool [isCopyNeeded](#) () const
Determine whether or not a copy is needed as determined by the referenced [ComputedExpression](#).
- [GarbageCollected](#) [makeCopy](#) () const
Create a separate copy of the original [GarbageCollected](#) value.
- [ComputedExpression](#) * operator-> () const
Access the tracked object as a pointer.
- [ComputedExpression](#) & operator* () const
Access the tracked object.
- bool operator== (const [Tang::integer_t](#) &val) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- bool operator== (const [Tang::float_t](#) &val) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- bool operator== (const bool &val) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- bool operator== (const std::string &val) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- bool operator== (const char *const &val) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- bool operator== (const [Error](#) &val) const
Compare the [GarbageCollected](#) tracked object with a supplied value.
- bool operator== (const std::nullptr_t &null) const
Compare the [GarbageCollected](#) tracked object with a supplied value.

- [GarbageCollected operator+](#) (const [GarbageCollected](#) &rhs) const
Perform an addition between two [GarbageCollected](#) values.
- [GarbageCollected operator-](#) (const [GarbageCollected](#) &rhs) const
Perform a subtraction between two [GarbageCollected](#) values.
- [GarbageCollected operator*](#) (const [GarbageCollected](#) &rhs) const
Perform a multiplication between two [GarbageCollected](#) values.
- [GarbageCollected operator/](#) (const [GarbageCollected](#) &rhs) const
Perform a division between two [GarbageCollected](#) values.
- [GarbageCollected operator%](#) (const [GarbageCollected](#) &rhs) const
Perform a modulo between two [GarbageCollected](#) values.
- [GarbageCollected operator-](#) () const
Perform a negation on the [GarbageCollected](#) value.
- [GarbageCollected operator!](#) () const
Perform a logical not on the [GarbageCollected](#) value.
- [GarbageCollected operator<](#) (const [GarbageCollected](#) &rhs) const
Perform a < between two [GarbageCollected](#) values.
- [GarbageCollected operator<=](#) (const [GarbageCollected](#) &rhs) const
Perform a <= between two [GarbageCollected](#) values.
- [GarbageCollected operator>](#) (const [GarbageCollected](#) &rhs) const
Perform a > between two [GarbageCollected](#) values.
- [GarbageCollected operator>=](#) (const [GarbageCollected](#) &rhs) const
Perform a >= between two [GarbageCollected](#) values.
- [GarbageCollected operator==](#) (const [GarbageCollected](#) &rhs) const
Perform a == between two [GarbageCollected](#) values.
- [GarbageCollected operator!=](#) (const [GarbageCollected](#) &rhs) const
Perform a != between two [GarbageCollected](#) values.

Static Public Member Functions

- `template<class T , typename... Args>`
static [GarbageCollected make](#) (Args... args)
Creates a garbage-collected object of the specified type.

Protected Member Functions

- [GarbageCollected](#) ()
Constructs a garbage-collected object of the specified type.

Protected Attributes

- `size_t * count`
The count of references to the tracked object.
- `ComputedExpression * ref`
A reference to the tracked object.
- `std::function< void(void)> recycle`
A cleanup function to recycle the object.

Friends

- `std::ostream & operator<< (std::ostream &out, const GarbageCollected &gc)`
Add friendly output.

5.38.1 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.

5.38.2 Constructor & Destructor Documentation

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

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

Copy Constructor.

Parameters

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

5.38.2.2 [GarbageCollected\(\)](#) [2/3]

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

Move Constructor.

Parameters

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

5.38.2.3 [~GarbageCollected\(\)](#)

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

Destructor.

Clean up the tracked object, if appropriate.

5.38.2.4 `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.
-----------------	---

5.38.3 Member Function Documentation

5.38.3.1 `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.

5.38.3.2 `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:

**5.38.3.3 makeCopy()**

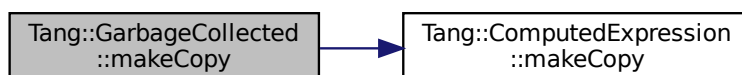
```
GarbageCollected GarbageCollected::makeCopy ( ) const
```

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:

**5.38.3.4 operator"!()**

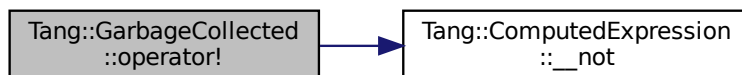
```
GarbageCollected GarbageCollected::operator! ( ) const
```

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

Returns

The result of the operation.

Here is the call graph for this function:

**5.38.3.5 operator"!=()**

```

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

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

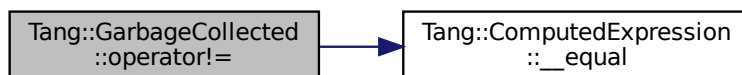
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:

**5.38.3.6 operator%()**

```

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

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

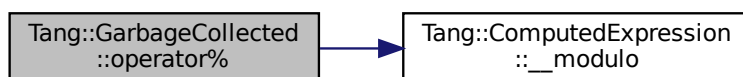
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.38.3.7 operator*() [1/2]

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

Access the tracked object.

Returns

A reference to the tracked object.

5.38.3.8 operator*() [2/2]

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

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

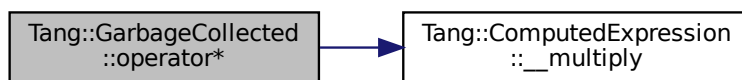
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.38.3.9 operator+()

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

Perform an addition between two `GarbageCollected` values.

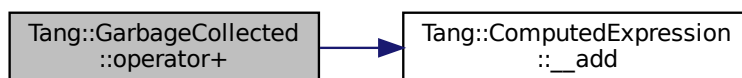
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.38.3.10 operator-() [1/2]

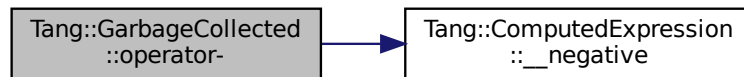
```
GarbageCollected GarbageCollected::operator- ( ) const
```

Perform a negation on the `GarbageCollected` value.

Returns

The result of the operation.

Here is the call graph for this function:

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

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

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

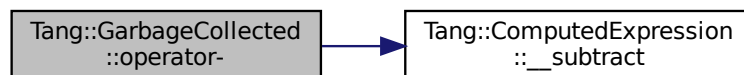
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:

**5.38.3.12 operator->()**

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

Access the tracked object as a pointer.

Returns

A pointer to the tracked object.

5.38.3.13 operator/()

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

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

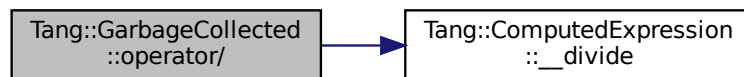
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:

**5.38.3.14 operator<()**

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

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

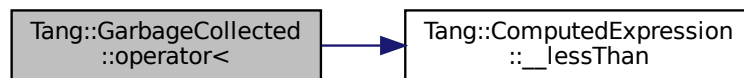
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.38.3.15 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.

5.38.3.16 operator=() [1/2]

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

Copy Assignment.

Parameters

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

5.38.3.17 operator=() [2/2]

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

Move Assignment.

Parameters

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

5.38.3.18 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.

5.38.3.19 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.

5.38.3.20 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.

5.38.3.21 operator==() [4/8]

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

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

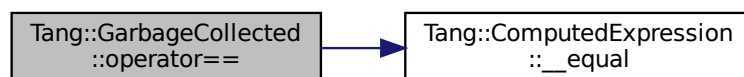
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:

**5.38.3.22 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.

5.38.3.23 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.

5.38.3.24 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.

5.38.3.25 operator==() [8/8]

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

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

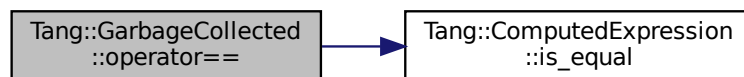
Parameters

<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:

5.38.3.26 `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.

5.38.3.27 `operator>=()`

```

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

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

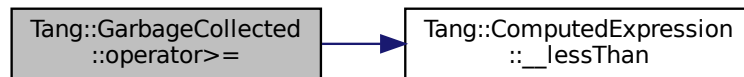
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.38.4 Friends And Related Function Documentation

5.38.4.1 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:

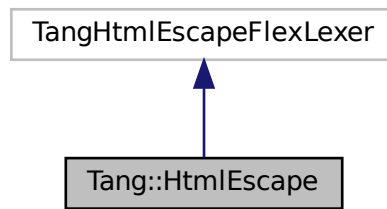
- [include/garbageCollected.hpp](#)
- [src/garbageCollected.cpp](#)

5.39 Tang::HtmlEscape Class Reference

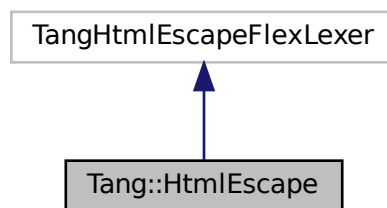
The Flex lexer class for the main Tang language.

```
#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.

5.39.1 Detailed Description

The Flex lexer class for the main Tang language.

Flex requires that our lexer class inherit from `yyFlexLexer`, an "intermediate" class whose real name is "`TangTang`↔`FlexLexer`". We are subclassing it so that we can override the return type of [get_next_token\(\)](#), for compatibility with Bison 3 tokens.

5.39.2 Constructor & Destructor Documentation

5.39.2.1 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)

5.39.3 Member Function Documentation

5.39.3.1 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:

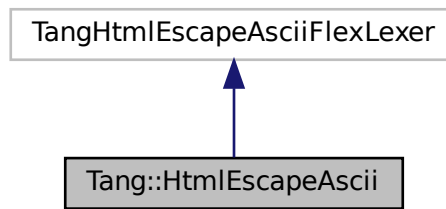
- [include/htmlEscape.hpp](#)

5.40 Tang::HtmlEscapeAscii Class Reference

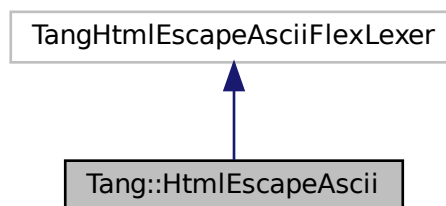
The Flex lexer class for the main Tang language.

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

Inheritance diagram for Tang::HtmlEscapeAscii:



Collaboration diagram for Tang::HtmlEscapeAscii:



Public Member Functions

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

5.40.1 Detailed Description

The Flex lexer class for the main Tang language.

Flex requires that our lexer class inherit from `yyFlexLexer`, an "intermediate" class whose real name is "`TangTang↔FlexLexer`". We are subclassing it so that we can override the return type of [get_next_token\(\)](#), for compatibility with Bison 3 tokens.

5.40.2 Constructor & Destructor Documentation

5.40.2.1 HtmlEscapeAscii()

```
Tang::HtmlEscapeAscii::HtmlEscapeAscii (
    std::istream & arg_yyin,
    std::ostream & arg_yyout ) [inline]
```

The constructor for the Scanner.

The design of the Flex lexer is to tokenize the contents of an input stream, and to write any error messages to an output stream. In our implementation, however, errors are returned differently, so the output stream is never used. It's presence is retained, however, in case it is needed in the future.

For now, the general approach should be to supply the input as a string stream, and to use std::cout as the output.

Parameters

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

5.40.3 Member Function Documentation

5.40.3.1 get_next_token()

```
virtual std::string Tang::HtmlEscapeAscii::get_next_token ( ) [virtual]
```

Extract the next token from the input string.

Returns

The next unescaped character.

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

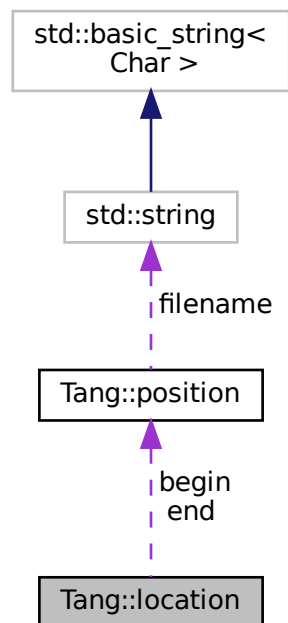
- include/[htmlEscapeAscii.hpp](#)

5.41 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.

5.41.1 Detailed Description

Two points in a source file.

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

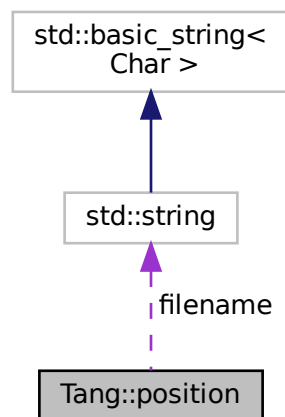
- build/generated/[location.hh](#)

5.42 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 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::integer_t](#))
Add a [Tang::integer_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::integer_t](#) [jumpTarget](#))
Set the target address of a Jump opcode.
- bool [setFunctionStackDeclaration](#) (size_t [opcodeAddress](#), [integer_t](#) [argc](#), [integer_t](#) [targetPC](#))
Set the stack details of a function declaration.
- void [pushEnvironment](#) (const std::shared_ptr< [AstNode](#) > &[ast](#))
Create a new compile/execute environment stack entry.
- void [popEnvironment](#) ()
Remove a compile/execute environment stack entry.
- void [addIdentifier](#) (const std::string &[name](#), std::optional< size_t > [position](#)={})
Add an identifier to the environment.
- const std::map< std::string, size_t > & [getIdentifiers](#) () const
Get the identifier map of the current environment.
- void [addIdentifierAssigned](#) (const std::string &[name](#))
Indicate that an identifier will be altered within the associated scope.
- const std::set< std::string > & [getIdentifiersAssigned](#) () const
Get the set of identifiers that will be assigned in the current scope.
- void [addString](#) (const std::string &[name](#))
Add a string to the environment.
- const std::map< std::string, size_t > & [getStrings](#) () const
Get the string map of the current environment.
- void [pushBreakStack](#) ()
*Increase the *break* environment stack, so that we can handle nested break-supporting structures.*
- void [addBreak](#) (size_t [location](#))
*Add the Bytecode location of a *break* statement, to be set when the final target is known at a later time.*
- void [popBreakStack](#) (size_t [target](#))
*For all *continue* bytecode locations collected by [Tang::addContinue](#), set the target pc to *target*.*
- void [pushContinueStack](#) ()

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

- void `addContinue` (size_t location)

Add the Bytecode location of a `continue` statement, to be set when the final target is known at a later time.

- void `popContinueStack` (size_t target)

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

Public Attributes

- std::string `out`

The output of the program, resulting from the program execution.

- std::vector< std::set< std::string > > `functionsCollected`

Names of the functions that are declared in a previous or the current scope.

- std::map< std::string, std::pair< uinteger_t, uinteger_t > > `functionsDeclared`

Key/value pair of the function declaration information.

- std::map< std::string, std::vector< Tang::uinteger_t > > `functionStackDeclarations`

For each function name, a list of Bytecode addresses that need to be replaced by a function definition.

Private Member Functions

- void `parse` ()

Parse the code into an AST.

- void `compile` ()

Compile the AST into Bytecode.

Private Attributes

- std::vector< std::map< std::string, size_t > > `identifierStack`

Stack of mappings of identifiers to their stack locations.

- std::vector< std::set< std::string > > `identifiersAssignedStack`

Stack of sets of identifiers that are the target of an assignment statement within the associated scope.

- std::vector< std::map< std::string, size_t > > `stringStack`

Stack of mappings of strings to their stack locations.

- std::vector< std::set< size_t > > `breakStack`

Stack of a collection of `break` statement locations.

- std::vector< std::set< size_t > > `continueStack`

Stack of a collection of `continue` statement locations.

- std::string `code`

The code supplied when the `Program` was instantiated.

- `CodeType` `codeType`

The type of code that was supplied when the `Program` was instantiated.

- shared_ptr< `AstNode` > `ast`

A pointer to the AST, if parsing was successful.

- `Bytecode` `bytecode`

The Bytecode of the compiled program.

- std::optional< `GarbageCollected` > `result`

The result of the `Program` execution.

5.43.1 Detailed Description

Represents a compiled script or template that may be executed.

5.43.2 Member Enumeration Documentation

5.43.2.1 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.

5.43.3 Constructor & Destructor Documentation

5.43.3.1 Program()

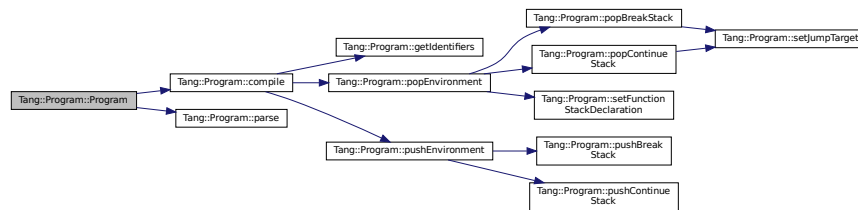
```
Program::Program (
    std::string code,
    Program::CodeType codeType )
```

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 <i>Script</i> or <i>Template</i> .

Here is the call graph for this function:



5.43.4 Member Function Documentation

5.43.4.1 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.
-----------------	---

5.43.4.2 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.

5.43.4.3 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.
-----------------	--

5.43.4.4 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.

5.43.4.5 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.
-------------	----------------------

5.43.4.6 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.

5.43.4.7 dumpBytecode()

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

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

Returns

A string containing the Opcode representation.

5.43.4.8 execute()

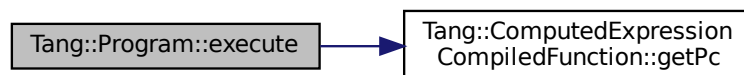
```
Program & Program::execute ( )
```

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

Returns

The current [Program](#) object.

Here is the call graph for this function:

**5.43.4.9 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.

5.43.4.10 getBytecode()

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

Get the Bytecode vector.

Returns

The Bytecode vector.

5.43.4.11 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.

5.43.4.12 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.

5.43.4.13 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.

5.43.4.14 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.

5.43.4.15 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.

5.43.4.16 popBreakStack()

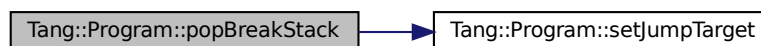
```
void Program::popBreakStack (
    size_t target )
```

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:



5.43.4.17 popContinueStack()

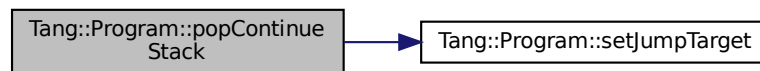
```
void Program::popContinueStack (
    size_t target )
```

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:



5.43.4.18 pushEnvironment()

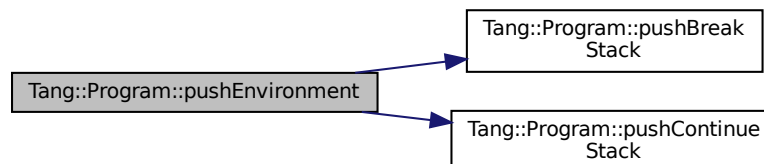
```
void Program::pushEnvironment (
    const std::shared_ptr< AstNode > & ast )
```

Create a new compile/execute environment stack entry.

Parameters

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

Here is the call graph for this function:



5.43.4.19 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.

5.43.4.20 setJumpTarget()

```
bool Program::setJumpTarget (
    size_t opcodeAddress,
    Tang::unsigned_t jumpTarget )
```

Set the target address of a Jump opcode.

Parameters

<i>opcodeAddress</i>	The location of the jump statement.
<i>jumpTarget</i>	The address to jump to.

Returns

Whether or not the jumpTarget was set.

5.43.5 Member Data Documentation

5.43.5.1 functionsDeclared

```
std::map<std::string, std::pair<unsigned_t, unsigned_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:

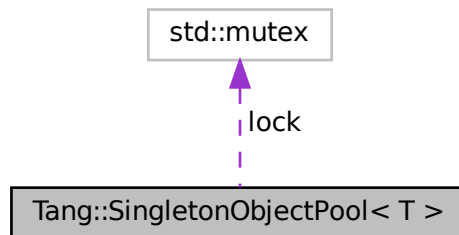
- [include/program.hpp](#)
- [src/program-dumpBytecode.cpp](#)
- [src/program-execute.cpp](#)
- [src/program.cpp](#)

5.44 Tang::SingletonObjectPool< T > Class Template Reference

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

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

Collaboration diagram for Tang::SingletonObjectPool< T >:



Public Member Functions

- `T * get ()`
Request an uninitialized memory location from the pool for an object T.
- `void recycle (T *obj)`
Recycle a memory location for an object T.
- `~SingletonObjectPool ()`
Destructor.

Static Public Member Functions

- `static SingletonObjectPool< T > & getInstance ()`
Get the singleton instance of the object pool.

Private Member Functions

- `SingletonObjectPool ()`
The constructor, hidden from being directly called.
- `SingletonObjectPool (const SingletonObjectPool &other)`
The copy constructor, hidden from being called.

Private Attributes

- `T** allocations`
C-array of allocated blocks, each block contains `GROW` objects.
- `int currentAllocation`
Index into `allocations`, representing the current block supplying non-recycled memory addresses.
- `size_t currentIndex`
Current location (within the most recently allocated block) of an available `T`.*
- `int currentRecycledAllocation`
Index into `allocations`, representing the current block tracking the recycled memory addresses.
- `int currentRecycledIndex`
Current location (within the `currentRecycledAllocation` block) of the last available `T`.*

Static Private Attributes

- `static std::mutex lock`
A mutex for thread-safety.

5.44.1 Detailed Description

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

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

5.44.2 Member Function Documentation

5.44.2.1 `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`.

5.44.2.2 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.

5.44.2.3 recycle()

```
template<class T >
void Tang::SingletonObjectPool< T >::recycle (
    T * obj ) [inline]
```

Recycle a memory location for an object T.

Parameters

<i>obj</i>	The memory location to recycle.
------------	---------------------------------

5.44.3 Member Data Documentation

5.44.3.1 currentIndex

```
template<class T >
size_t Tang::SingletonObjectPool< T >::currentIndex [private]
```

Current location (within the most recently allocated block) of an available T*.

If currentIndex == GROW, then a new block needs to be allocated.

5.44.3.2 currentRecycledIndex

```
template<class T >
int Tang::SingletonObjectPool< T >::currentRecycledIndex [private]
```

Current location (within the currentRecycledAllocation block) of the last available T*.

If currentRecycledIndex == GROW, then we must move to the next currentRecycledAllocation.

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

- include/[singletonObjectPool.hpp](#)

5.45 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](#).

5.45.1 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).

5.45.2 Constructor & Destructor Documentation

5.45.2.1 TangBase()

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

The constructor.

Isn't it glorious.

5.45.3 Member Function Documentation

5.45.3.1 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:

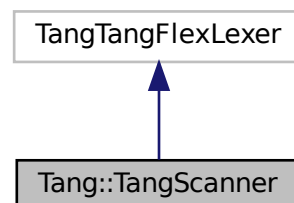
- include/[tangBase.hpp](#)
- src/[tangBase.cpp](#)

5.46 Tang::TangScanner Class Reference

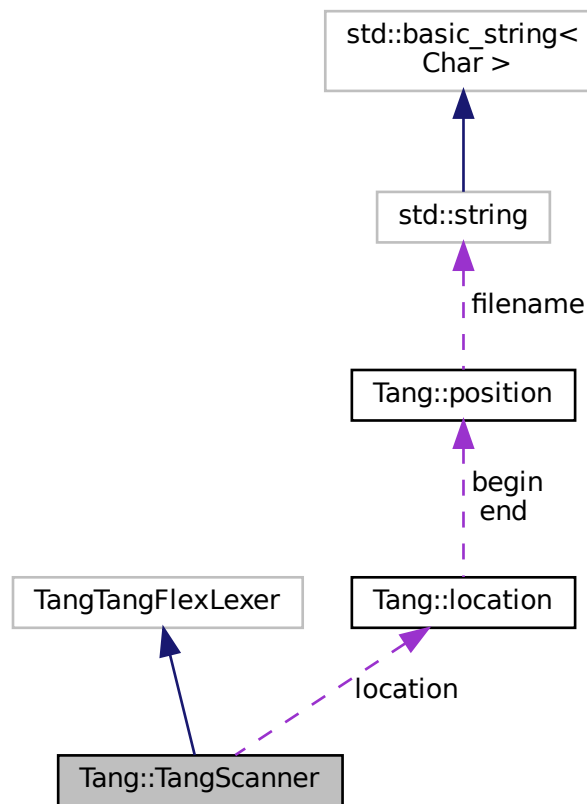
The Flex lexer class for the main Tang language.

```
#include <tangScanner.hpp>
```

Inheritance diagram for Tang::TangScanner:



Collaboration diagram for Tang::TangScanner:



Public Member Functions

- [TangScanner](#) (std::istream &arg_yyin, std::ostream &arg_yyout)
The constructor for the Scanner.
- virtual Tang::TangParser::symbol_type [get_next_token](#) ()
A pass-through function that we supply so that we can provide a Bison 3 token return type instead of the `int` that is returned by the default class configuration.

Private Attributes

- [Tang::location](#) `location`
The location information of the token that is identified.

5.46.1 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.

5.46.2 Constructor & Destructor Documentation

5.46.2.1 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)

5.46.3 Member Function Documentation

5.46.3.1 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:

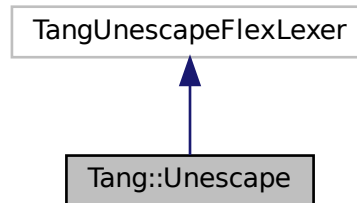
- include/[tangScanner.hpp](#)

5.47 Tang::Unescape Class Reference

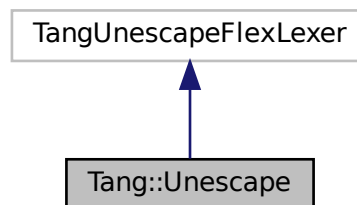
The Flex lexer class for the main Tang language.

```
#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.

5.47.1 Detailed Description

The Flex lexer class for the main Tang language.

Flex requires that our lexer class inherit from yyFlexLexer, an "intermediate" class whose real name is "TangTang↔ FlexLexer". We are subclassing it so that we can override the return type of [get_next_token\(\)](#), for compatibility with Bison 3 tokens.

5.47.2 Constructor & Destructor Documentation

5.47.2.1 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)

5.47.3 Member Function Documentation

5.47.3.1 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:

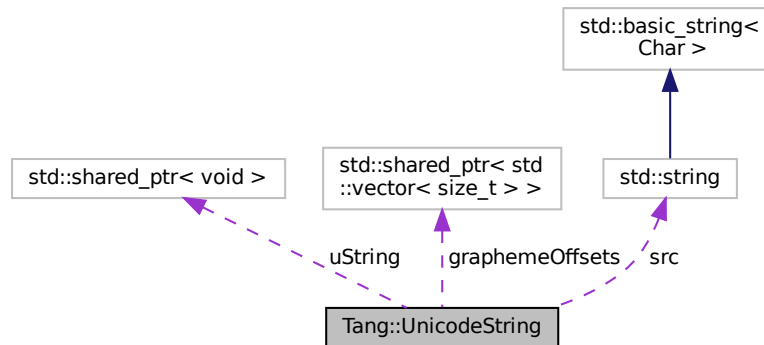
- include/[unescape.hpp](#)

5.48 Tang::UnicodeString Class Reference

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

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

Collaboration diagram for Tang::UnicodeString:



Public Member Functions

- `UnicodeString` (const std::string &src)
Construct a *Tang::UnicodeString* object, which acts as the interface to the ICU library.
- std::string substr (size_t position, size_t length) const
Return a Unicode grapheme-aware substring.
- bool operator== (const UnicodeString &rhs) const
Compare two 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.

Private Member Functions

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

Private Attributes

- `std::string` [src](#)
The UTF-8 encoded string.
- `std::shared_ptr< std::vector< size_t > >` [graphemeOffsets](#)
Cache of the grapheme offsets, if they happen to be calculated.
- `std::shared_ptr< void >` [uString](#)
Cache of the ICU Unicode string.

5.48.1 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.

5.48.2 Constructor & Destructor Documentation

5.48.2.1 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.
------------	-------------------------

5.48.3 Member Function Documentation

5.48.3.1 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.

5.48.3.2 length()

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

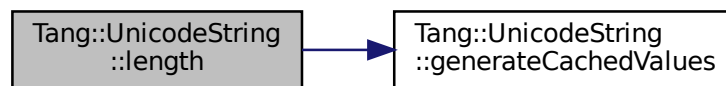
Return the length of the [UnicodeString](#) in graphemes.

Note: this is *not* the number of bytes, chars, or codepoints, but is the length in graphemes, as defined by ICU.

Returns

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

Here is the call graph for this function:



5.48.3.3 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](#).

5.48.3.4 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.

5.48.3.5 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.

5.48.3.6 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.

5.48.3.7 substr()

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

Return a Unicode grapheme-aware substring.

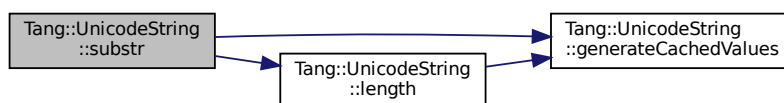
Parameters

<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

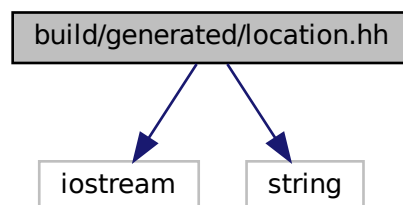
File Documentation

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.

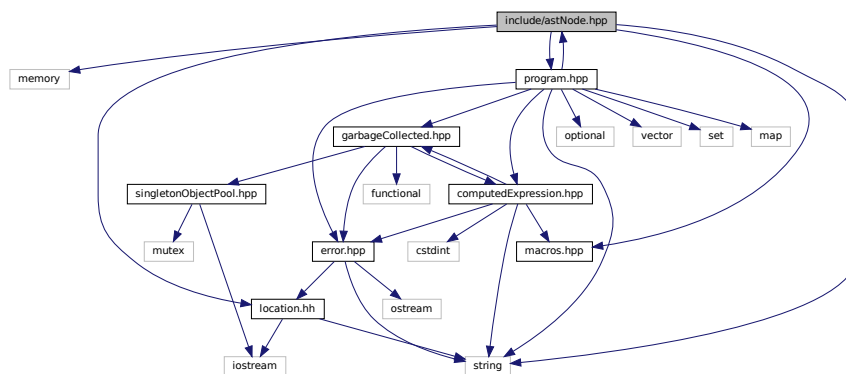
Parameters

<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).

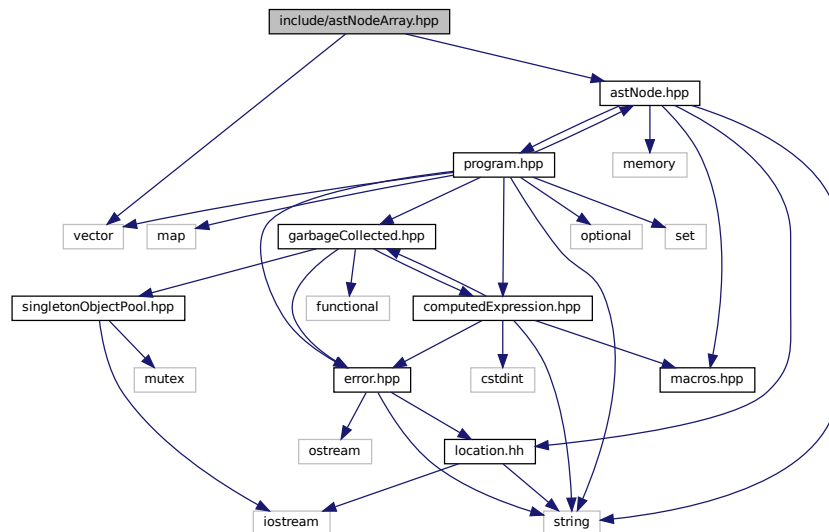
6.2.1 Detailed Description

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

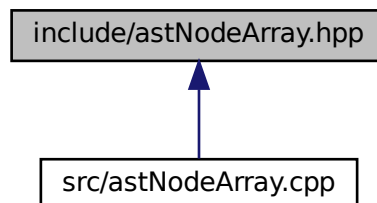
6.3 include/astNodeArray.hpp File Reference

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.

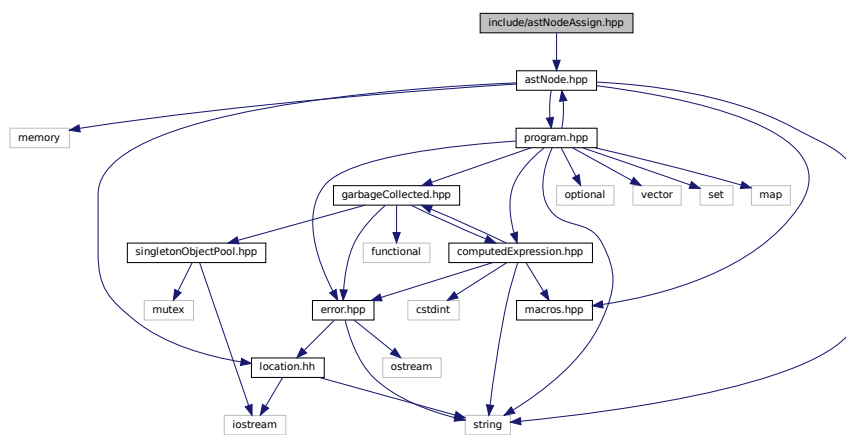
6.3.1 Detailed Description

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

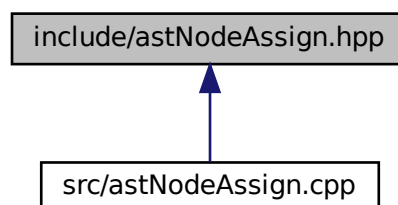
6.4 include/astNodeAssign.hpp File Reference

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

6.5.1 Detailed Description

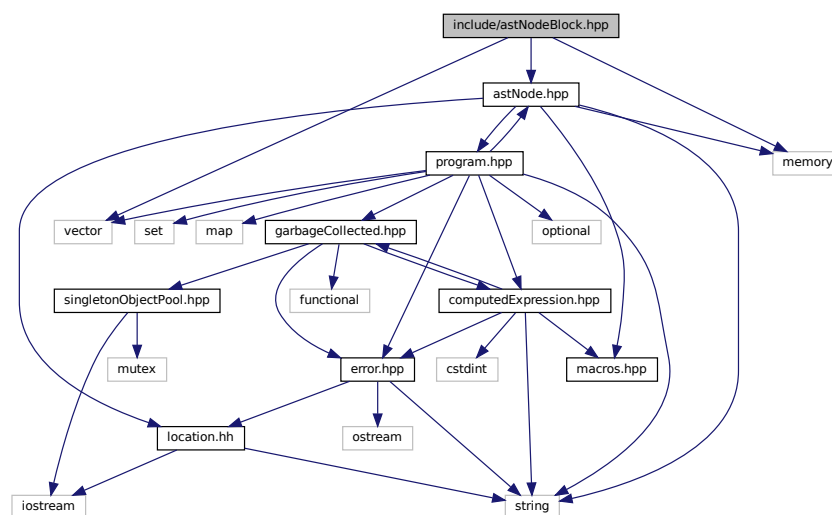
Declare the [Tang::AstNodeBinary](#) class.

6.6 include/astNodeBlock.hpp File Reference

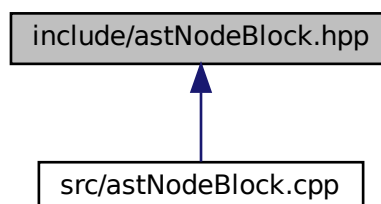
Declare the [Tang::AstNodeBlock](#) class.

```
#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

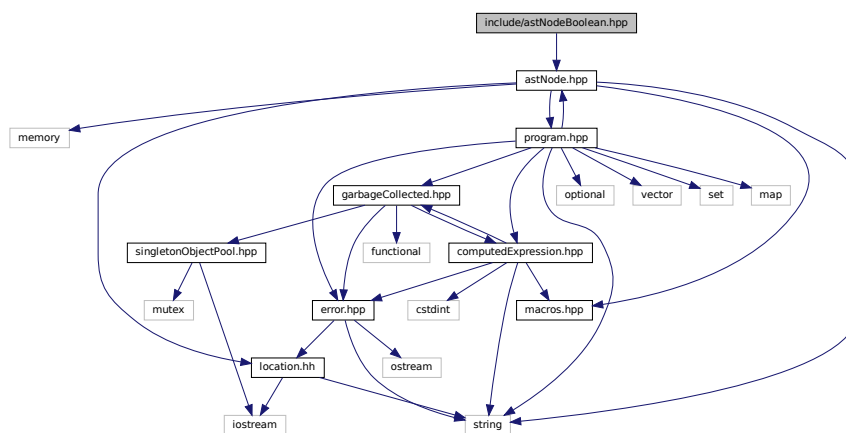
Declare the [Tang::AstNodeBlock](#) class.

6.7 include/astNodeBoolean.hpp File Reference

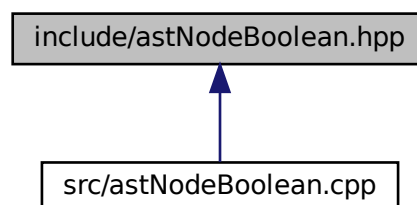
Declare the [Tang::AstNodeBoolean](#) class.

```
#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.

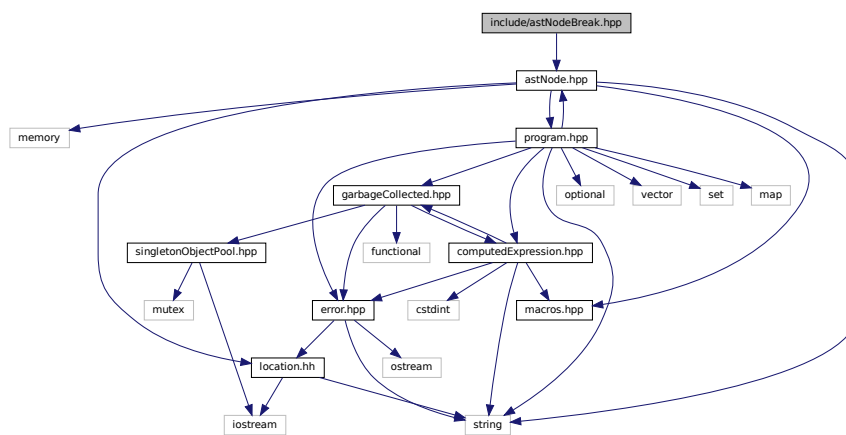
6.7.1 Detailed Description

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

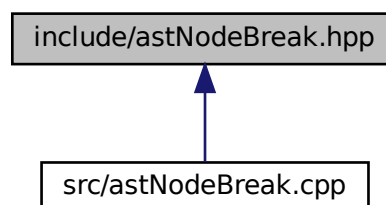
6.8 include/astNodeBreak.hpp File Reference

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.

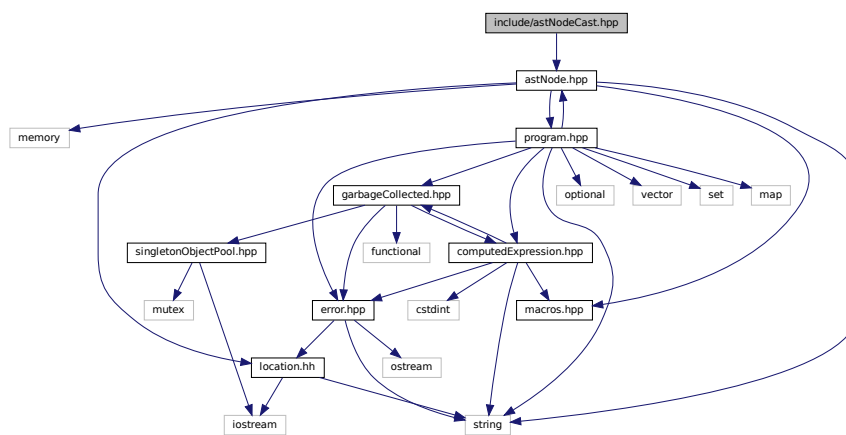
6.8.1 Detailed Description

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

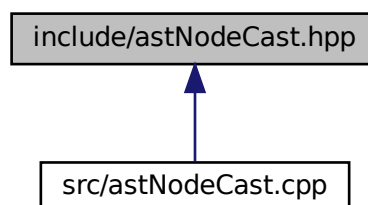
6.9 include/astNodeCast.hpp File Reference

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

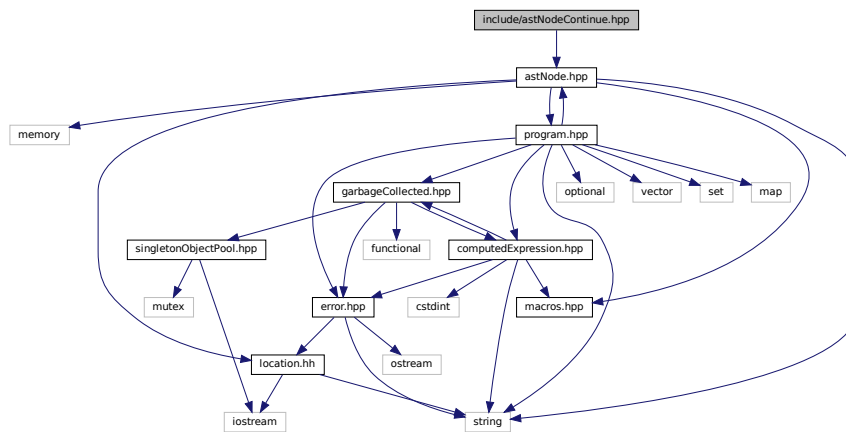
Declare the [Tang::AstNodeCast](#) class.

6.10 include/astNodeContinue.hpp File Reference

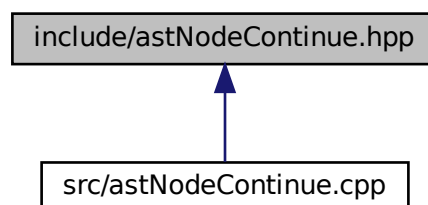
Declare the [Tang::AstNodeContinue](#) class.

```
#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

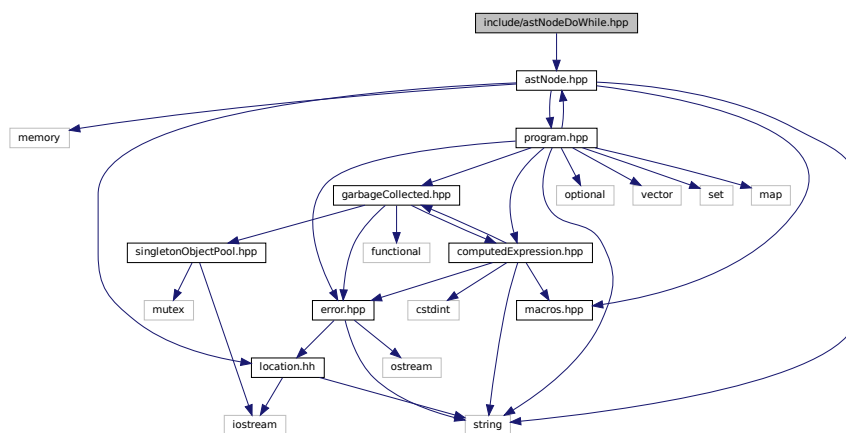
Declare the [Tang::AstNodeContinue](#) class.

6.11 include/astNodeDoWhile.hpp File Reference

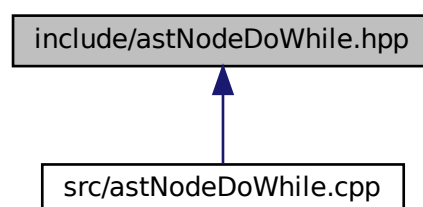
Declare the [Tang::AstNodeDoWhile](#) class.

```
#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.

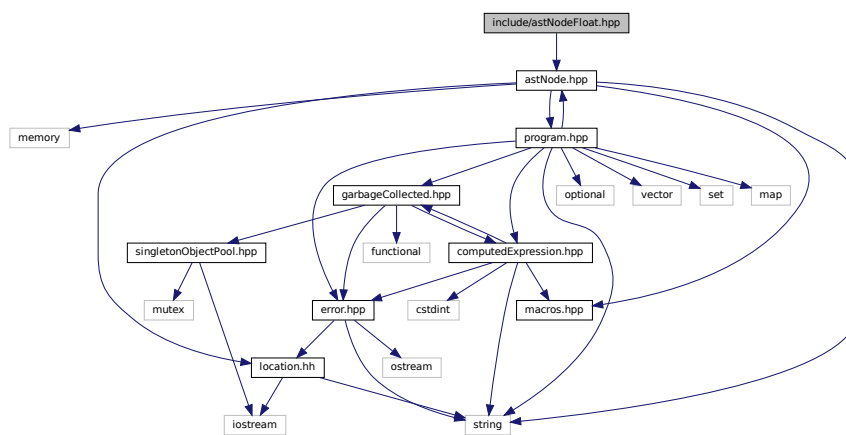
6.11.1 Detailed Description

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

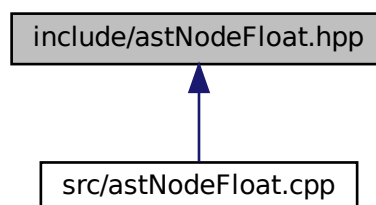
6.12 include/astNodeFloat.hpp File Reference

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.

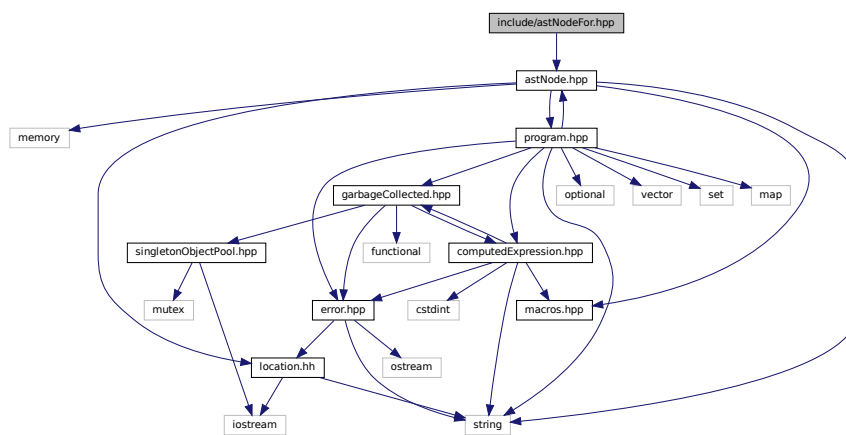
6.12.1 Detailed Description

Declare the [Tang::AstNodeFloat](#) class.

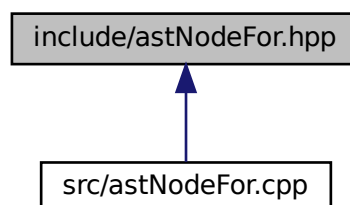
6.13 include/astNodeFor.hpp File Reference

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

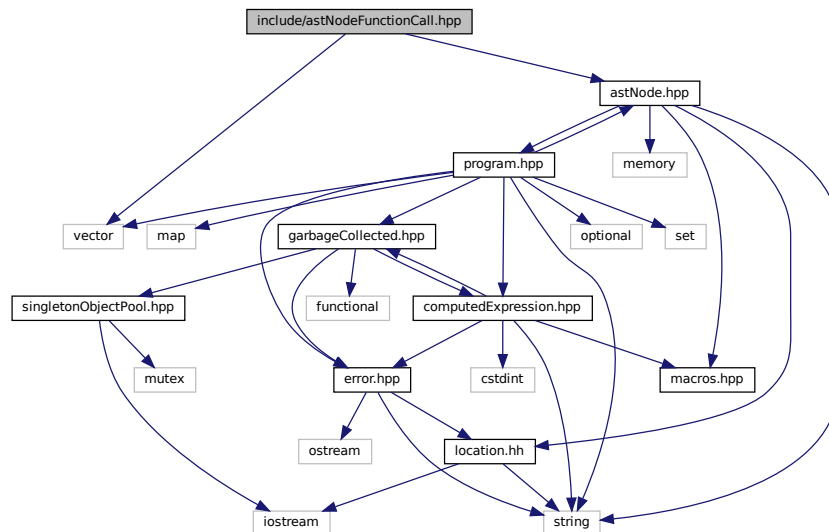
Declare the [Tang::AstNodeFor](#) class.

6.14 include/astNodeFunctionCall.hpp File Reference

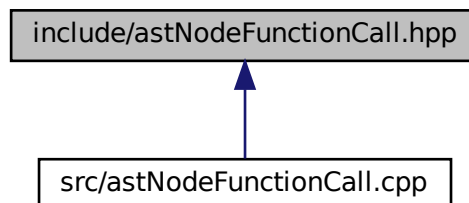
Declare the [Tang::AstNodeFunctionCall](#) class.

```
#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.

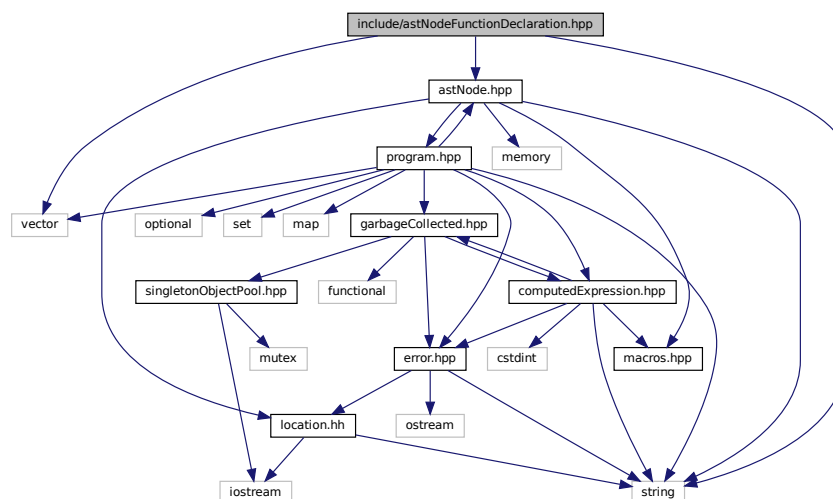
6.14.1 Detailed Description

Declare the [Tang::AstNodeFunctionCall](#) class.

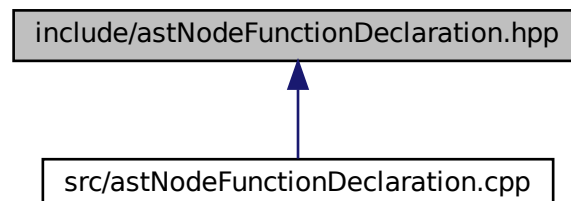
6.15 include/astNodeFunctionDeclaration.hpp File Reference

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.

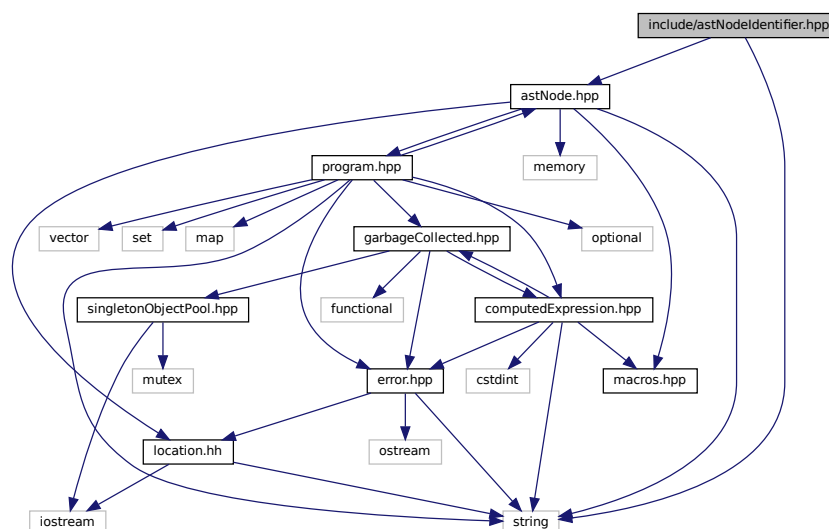
6.15.1 Detailed Description

Declare the [Tang::AstNodeFunctionDeclaration](#) class.

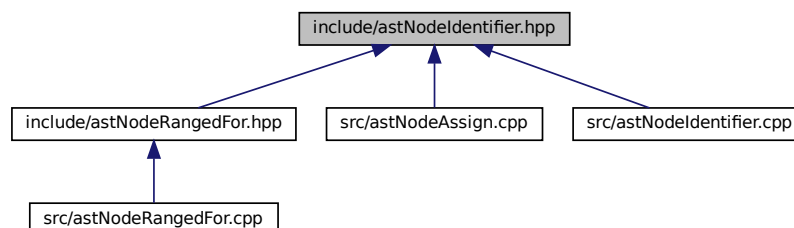
6.16 include/astNodeIdentifier.hpp File Reference

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.

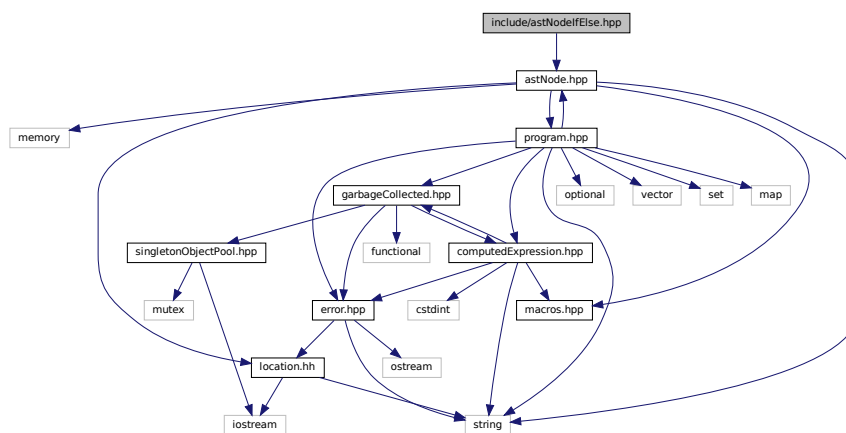
6.16.1 Detailed Description

Declare the [Tang::AstNodeIdentifier](#) class.

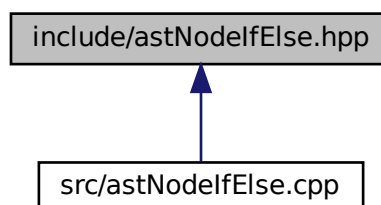
6.17 include/astNodeIfElse.hpp File Reference

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.

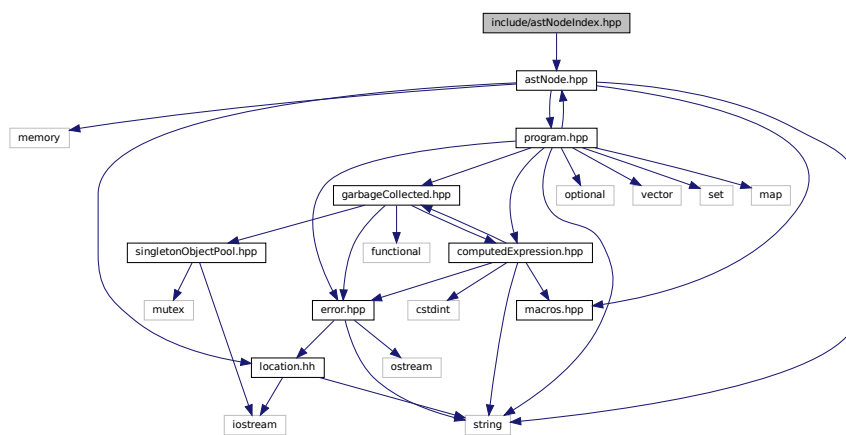
6.17.1 Detailed Description

Declare the [Tang::AstNodeIfElse](#) class.

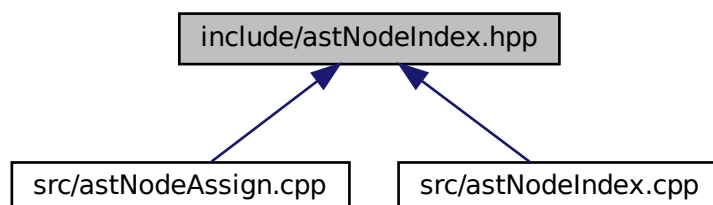
6.18 include/astNodeIndex.hpp File Reference

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

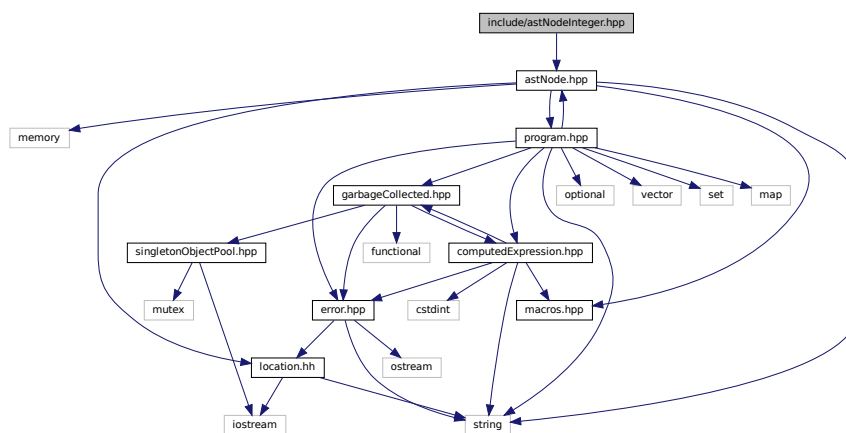
Declare the [Tang::AstNodeIndex](#) class.

6.19 include/astNodeInteger.hpp File Reference

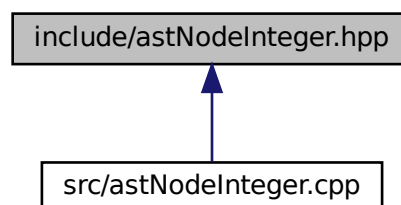
Declare the [Tang::AstNodeInteger](#) class.

```
#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.

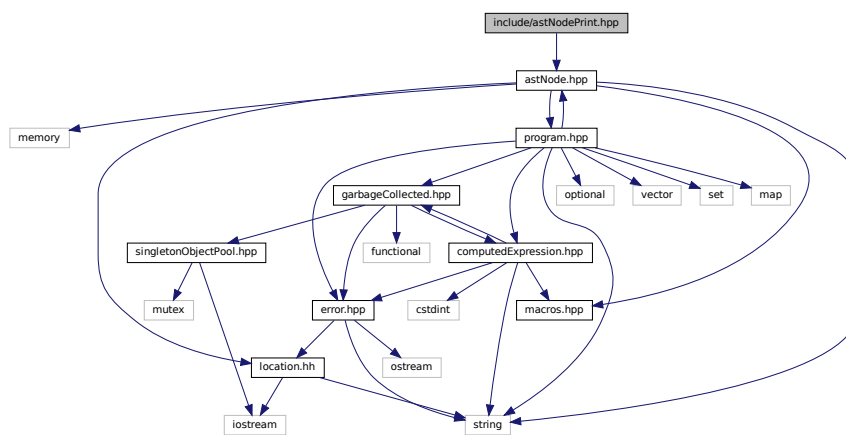
6.19.1 Detailed Description

Declare the [Tang::AstNodeInteger](#) class.

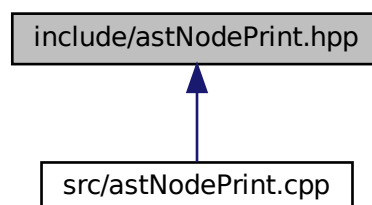
6.20 include/astNodePrint.hpp File Reference

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.

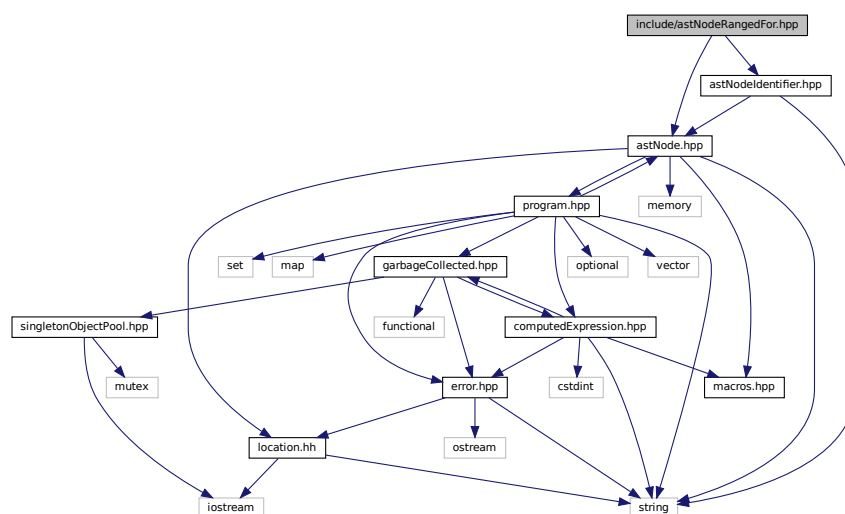
6.20.1 Detailed Description

Declare the [Tang::AstNodePrint](#) class.

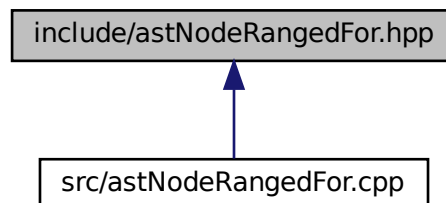
6.21 include/astNodeRangedFor.hpp File Reference

Declare the [Tang::AstNodeRangedFor](#) class.

```
#include "astNode.hpp"
#include "astNodeIdentifier.hpp"
Include dependency graph for astNodeRangedFor.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeRangedFor](#)
An *AstNode* that represents a ranged for() statement.

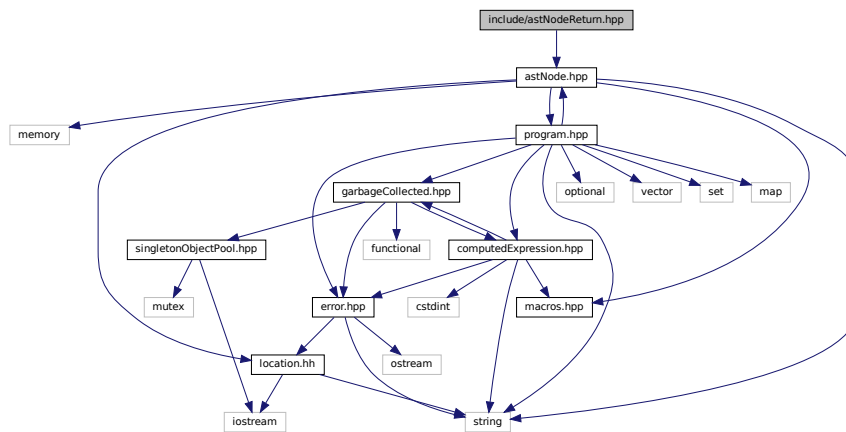
6.21.1 Detailed Description

Declare the [Tang::AstNodeRangedFor](#) class.

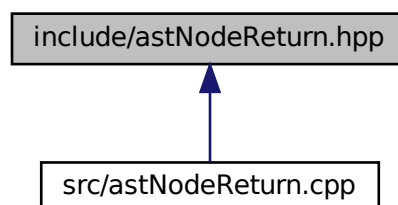
6.22 include/astNodeReturn.hpp File Reference

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.

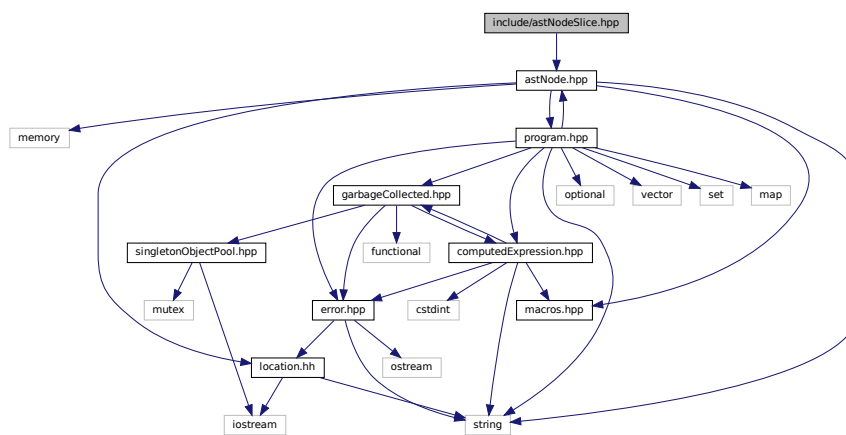
6.22.1 Detailed Description

Declare the [Tang::AstNodeReturn](#) class.

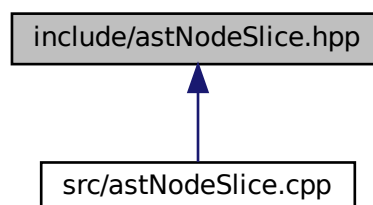
6.23 include/astNodeSlice.hpp File Reference

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.

6.23.1 Detailed Description

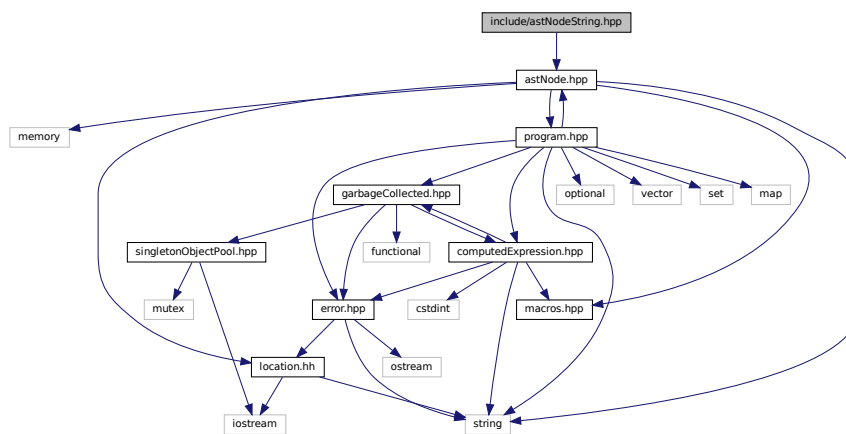
Declare the [Tang::AstNodeSlice](#) class.

6.24 include/astNodeString.hpp File Reference

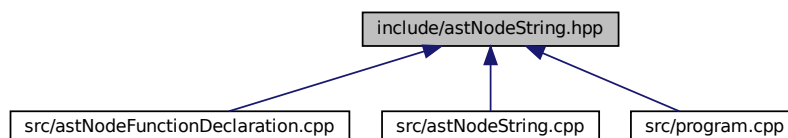
Declare the [Tang::AstNodeString](#) class.

```
#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.

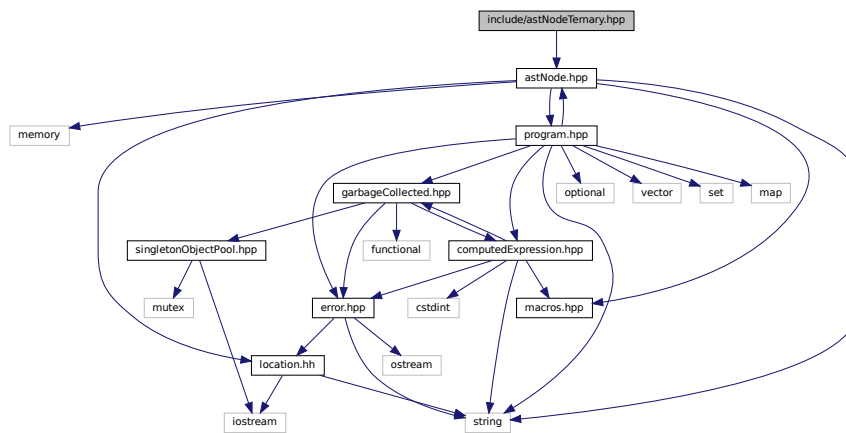
6.24.1 Detailed Description

Declare the [Tang::AstNodeString](#) class.

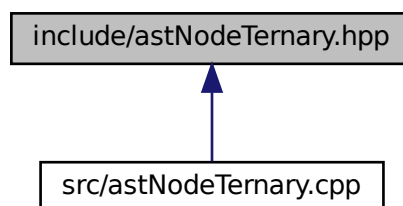
6.25 include/astNodeTernary.hpp File Reference

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.

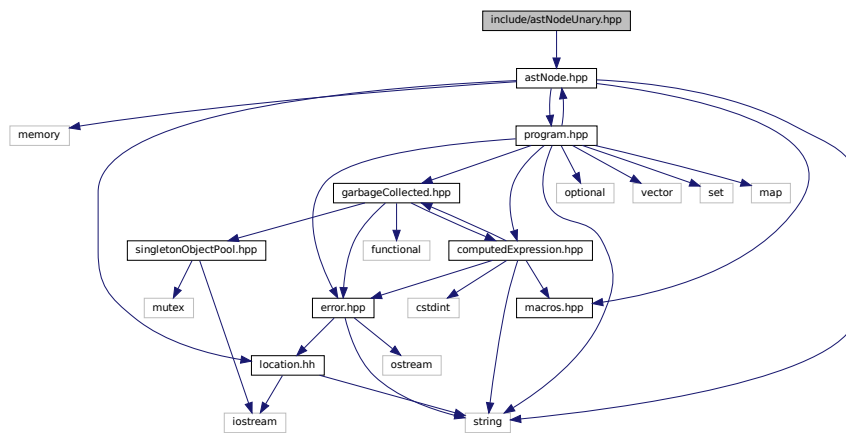
6.25.1 Detailed Description

Declare the [Tang::AstNodeUnary](#) class.

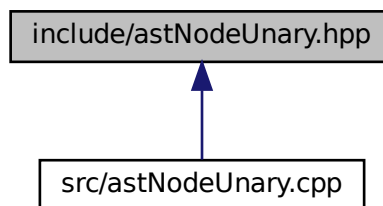
6.26 include/astNodeUnary.hpp File Reference

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.

6.26.1 Detailed Description

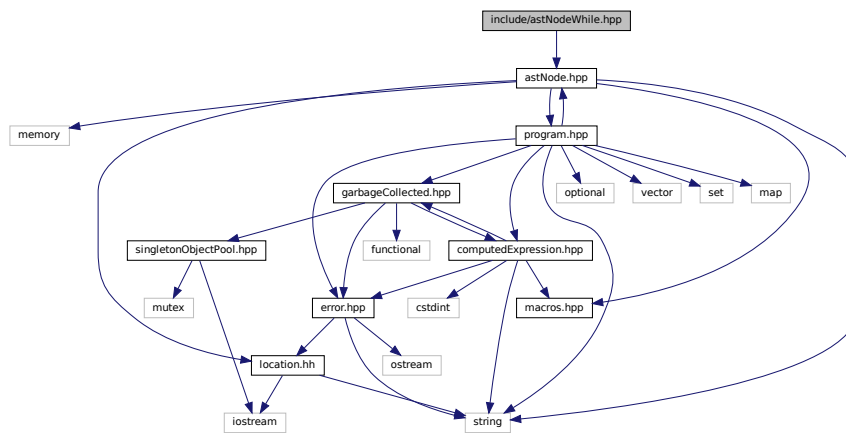
Declare the [Tang::AstNodeUnary](#) class.

6.27 include/astNodeWhile.hpp File Reference

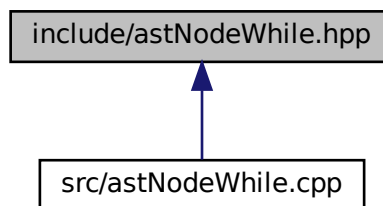
Declare the [Tang::AstNodeWhile](#) class.

```
#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.

6.27.1 Detailed Description

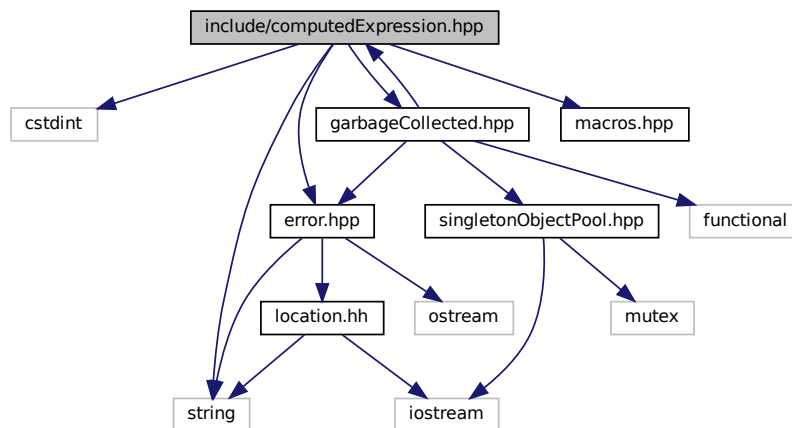
Declare the `Tang::AstNodeWhile` class.

6.28 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.

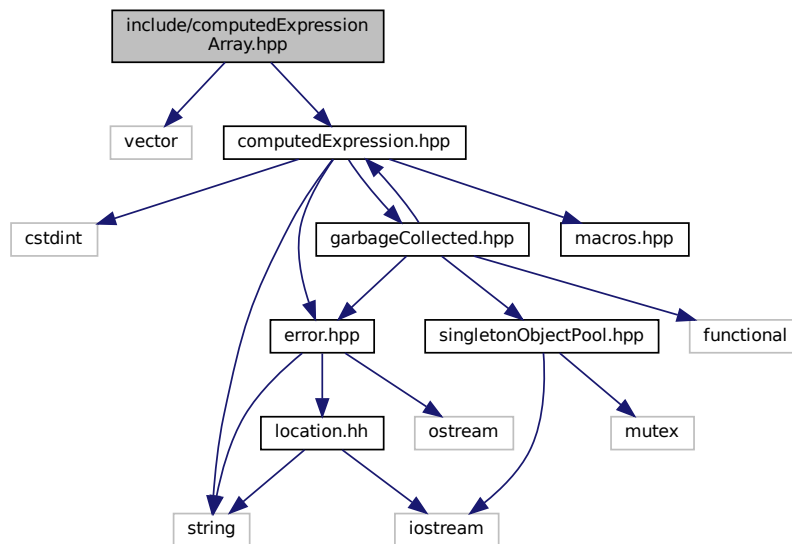
6.28.1 Detailed Description

Declare the `Tang::ComputedExpression` base class.

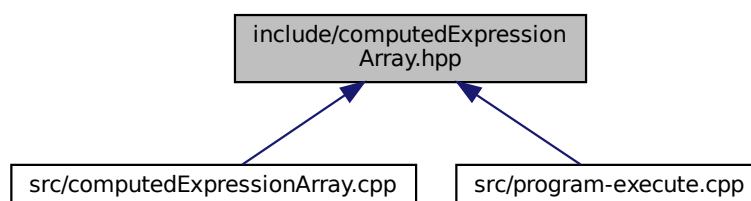
6.29 include/computedExpressionArray.hpp File Reference

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.

6.29.1 Detailed Description

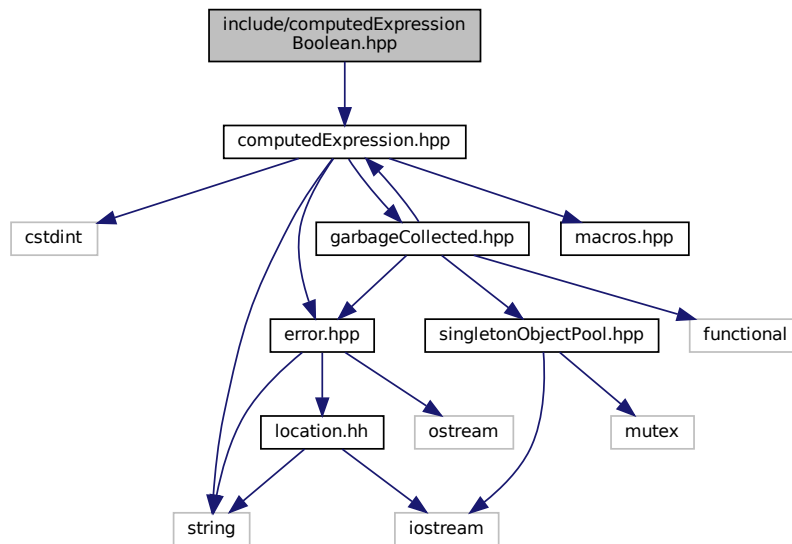
Declare the [Tang::ComputedExpressionArray](#) class.

6.30 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.

6.30.1 Detailed Description

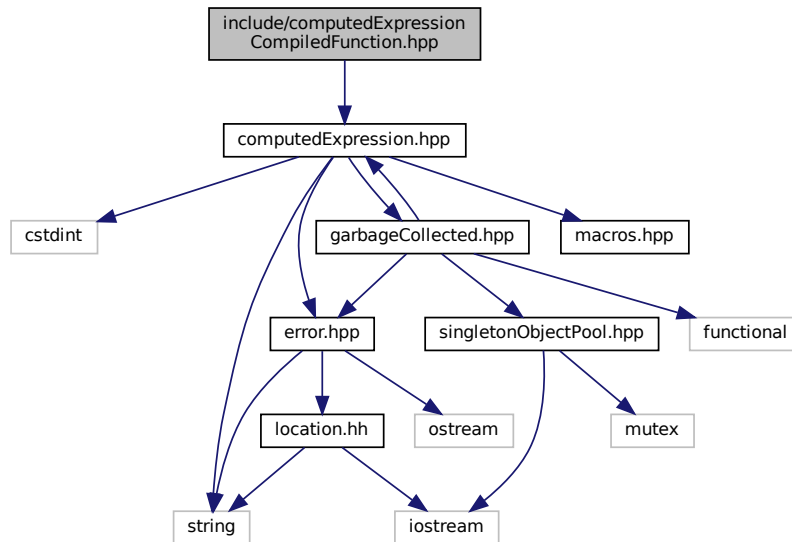
Declare the [Tang::ComputedExpressionBoolean](#) class.

6.31 include/computedExpressionCompiledFunction.hpp File Reference

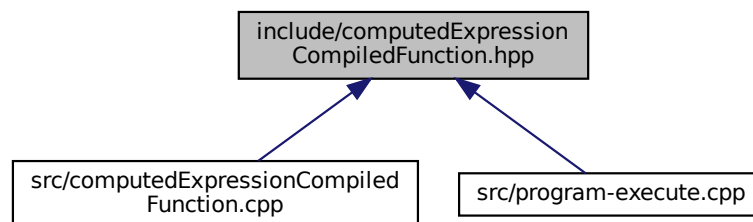
Declare the [Tang::ComputedExpressionCompiledFunction](#) class.

```
#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.

6.31.1 Detailed Description

Declare the [Tang::ComputedExpressionCompiledFunction](#) class.

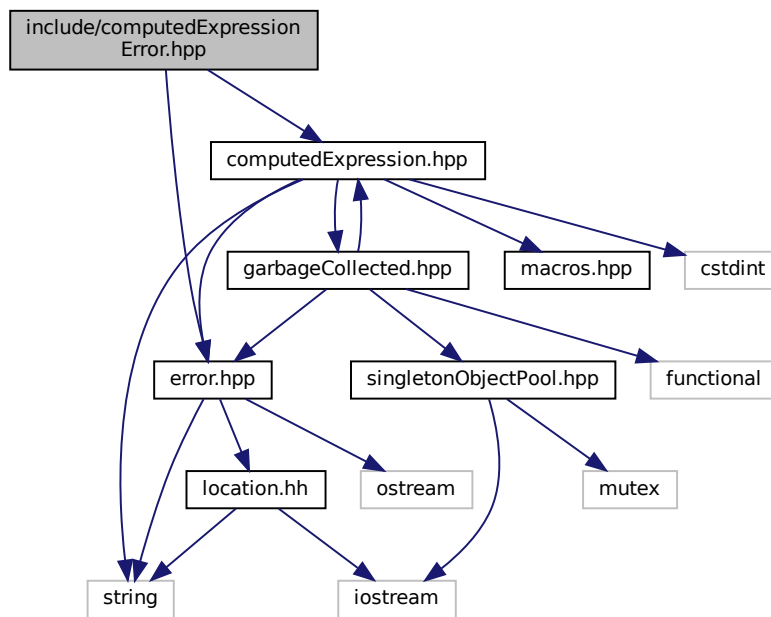
6.32 include/computedExpressionError.hpp File Reference

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](#).

6.32.1 Detailed Description

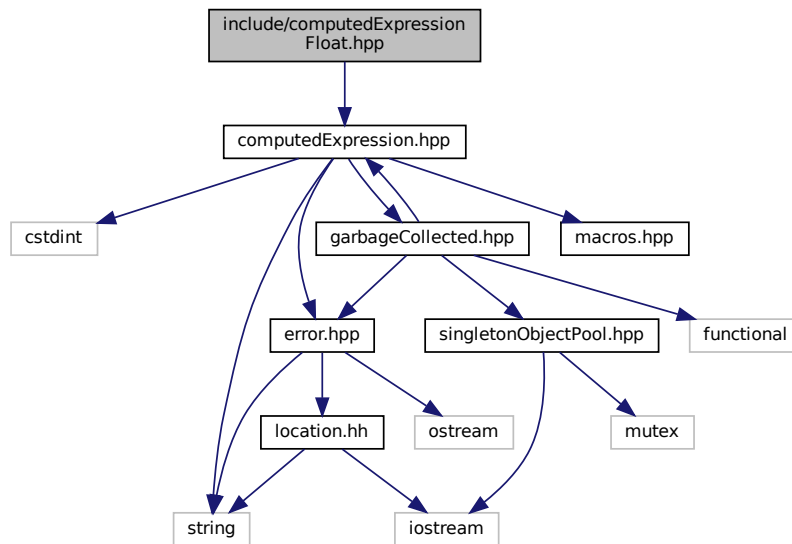
Declare the [Tang::ComputedExpressionError](#) class.

6.33 include/computedExpressionFloat.hpp File Reference

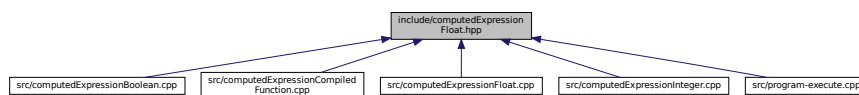
Declare the [Tang::ComputedExpressionFloat](#) class.

```
#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.

6.33.1 Detailed Description

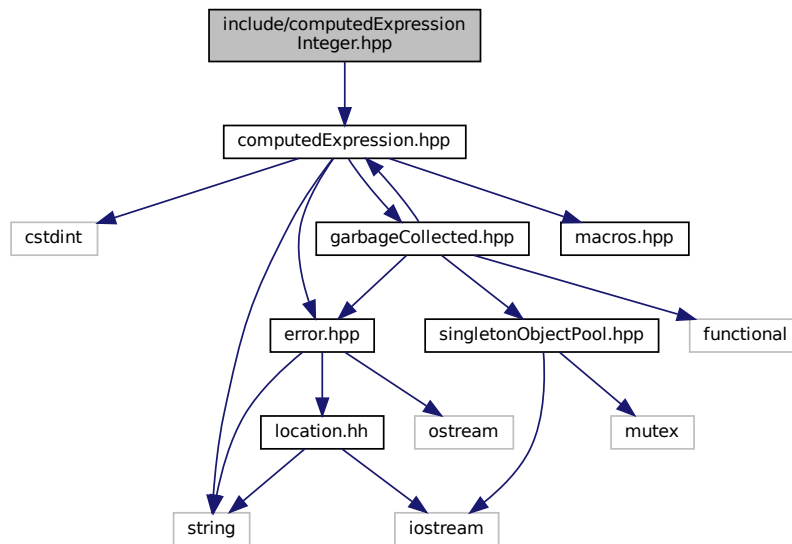
Declare the [Tang::ComputedExpressionFloat](#) class.

6.34 include/computedExpressionInteger.hpp File Reference

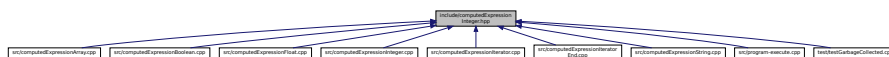
Declare the [Tang::ComputedExpressionInteger](#) class.

```
#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.

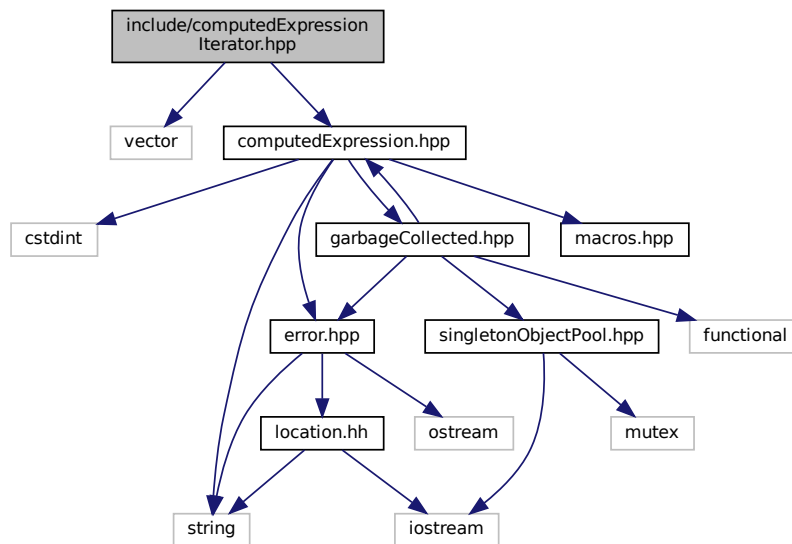
6.34.1 Detailed Description

Declare the [Tang::ComputedExpressionInteger](#) class.

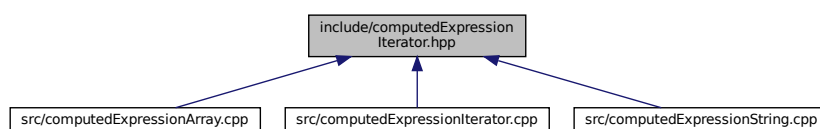
6.35 include/computedExpressionIterator.hpp File Reference

Declare the [Tang::ComputedExpressionIterator](#) class.

```
#include <vector>
#include "computedExpression.hpp"
Include dependency graph for computedExpressionIterator.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionIterator](#)
Represents an Iterator that is the result of a computation.

6.35.1 Detailed Description

Declare the [Tang::ComputedExpressionIterator](#) class.

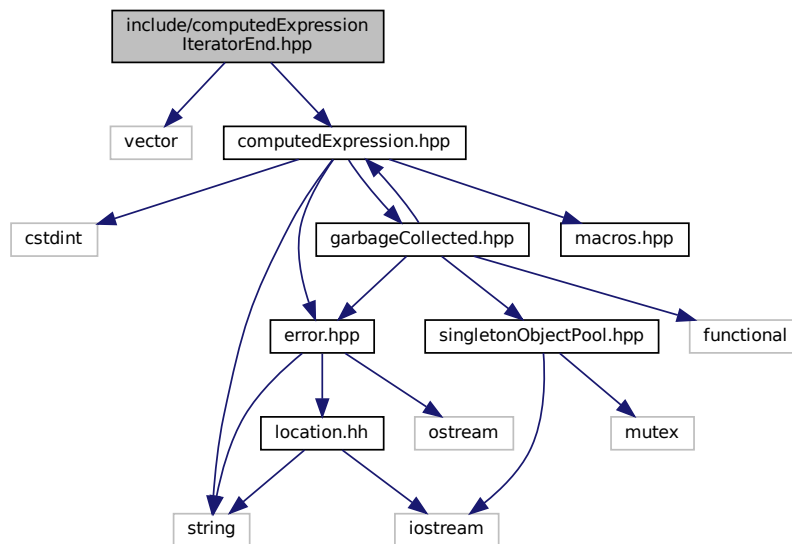
6.36 include/computedExpressionIteratorEnd.hpp File Reference

Declare the [Tang::ComputedExpressionIteratorEnd](#) class.

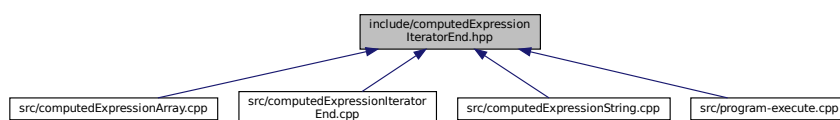
```
#include <vector>
```

```
#include "computedExpression.hpp"
```

Include dependency graph for computedExpressionIteratorEnd.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionIteratorEnd](#)

Represents that a collection has no more values through which to iterate.

6.36.1 Detailed Description

Declare the [Tang::ComputedExpressionIteratorEnd](#) class.

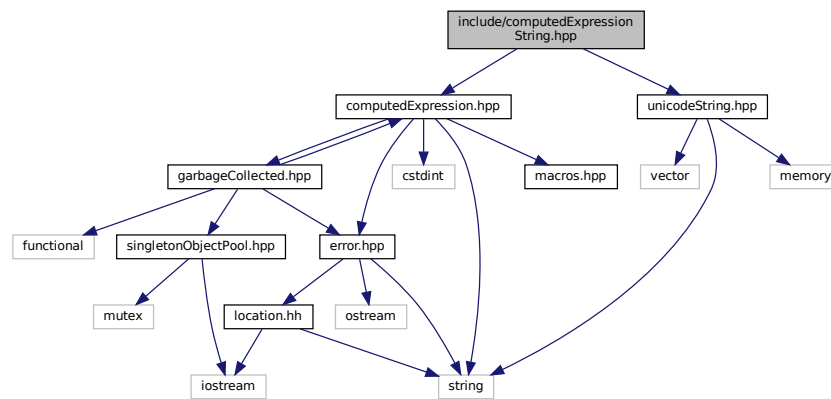
6.37 include/computedExpressionString.hpp File Reference

Declare the [Tang::ComputedExpressionString](#) class.

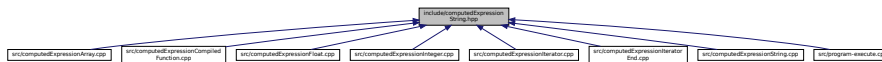
```
#include "computedExpression.hpp"
```

```
#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.

6.37.1 Detailed Description

Declare the [Tang::ComputedExpressionString](#) class.

6.38 include/error.hpp File Reference

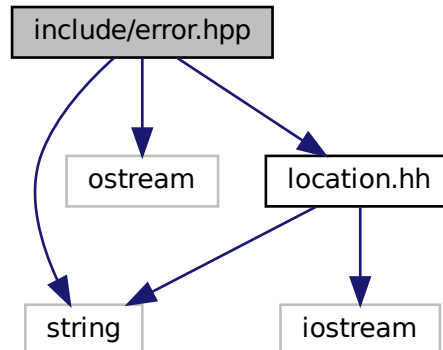
Declare the [Tang::Error](#) class used to describe syntax and runtime errors.

```
#include <string>
```

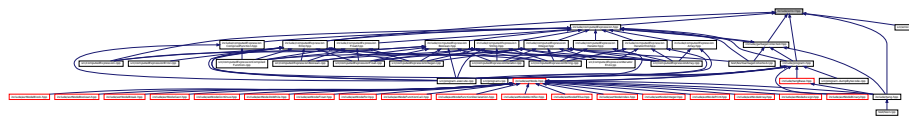
```
#include <ostream>
```

```
#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.

6.38.1 Detailed Description

Declare the [Tang::Error](#) class used to describe syntax and runtime errors.

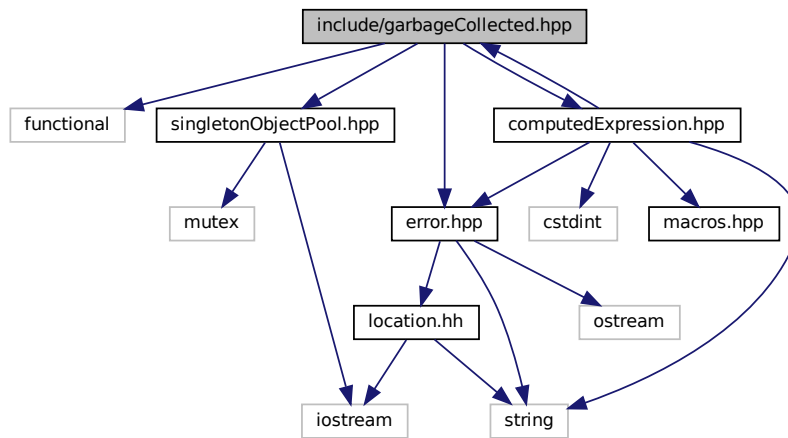
6.39 include/garbageCollected.hpp File Reference

Declare the [Tang::GarbageCollected](#) class.

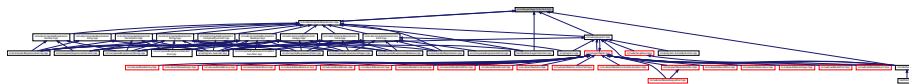
```
#include <functional>
#include "singletonObjectPool.hpp"
#include "computedExpression.hpp"
```

```
#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.

6.39.1 Detailed Description

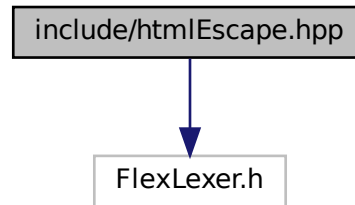
Declare the [Tang::GarbageCollected](#) class.

6.40 include/htmlEscape.hpp File Reference

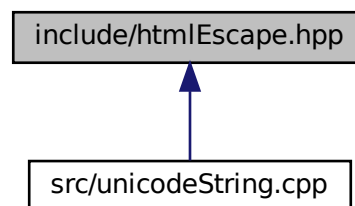
Declare the [Tang::HtmlEscape](#) used to tokenize a Tang script.

```
#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\(\)](#)

6.40.1 Detailed Description

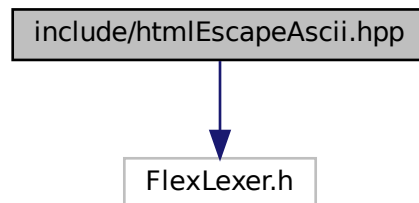
Declare the [Tang::HtmlEscape](#) used to tokenize a Tang script.

6.41 include/htmlEscapeAscii.hpp File Reference

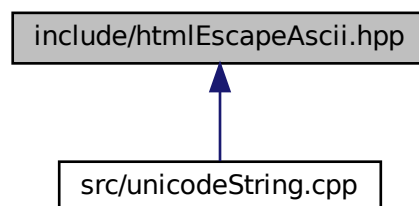
Declare the [Tang::HtmlEscapeAscii](#) used to tokenize a Tang script.

```
#include <FlexLexer.h>
```

Include dependency graph for htmlEscapeAscii.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::HtmlEscapeAscii](#)
The Flex lexer class for the main Tang language.

Macros

- #define **yyFlexLexer** TangHtmlEscapeAsciiFlexLexer
- #define **YY_DECL** std::string [Tang::HtmlEscapeAscii::get_next_token\(\)](#)

6.41.1 Detailed Description

Declare the [Tang::HtmlEscapeAscii](#) used to tokenize a Tang script.

6.42 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.

6.42.1 Detailed Description

Contains generic macros.

6.43 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](#) ,
[GETITERATOR](#) , [ITERATORNEXT](#) , [ISITERATOREND](#) , [CASTINTEGER](#) ,
[CASTFLOAT](#) , [CASTBOOLEAN](#) , [CALLFUNC](#) , [RETURN](#) ,
[PRINT](#) }

6.43.1 Detailed Description

Declare the Opcodes used in the Bytecode representation of a program.

6.43.2 Enumeration Type Documentation

6.43.2.1 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].
GETITERATOR	Pop a collection, push the collection iterator.

Enumerator

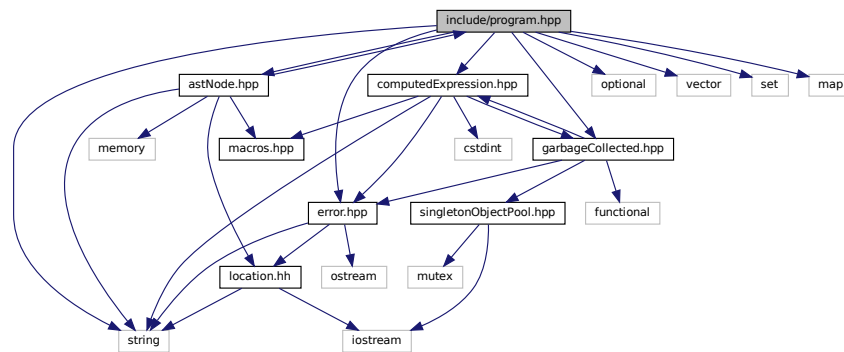
ITERATORNEXT	Pop an iterator, push the next iterator value.
ISITERATOREND	Pop a val, push bool(is val == iterator end)
CASTINTEGER	Pop a val, typecast to int, push.
CASTFLOAT	Pop a val, typecast to float, push.
CASTBOOLEAN	Pop a val, typecast to boolean, push.
CALLFUNC	Get argc, Pop a function, execute 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.

6.44 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.

6.44.1 Detailed Description

Declare the [Tang::Program](#) class used to compile and execute source code.

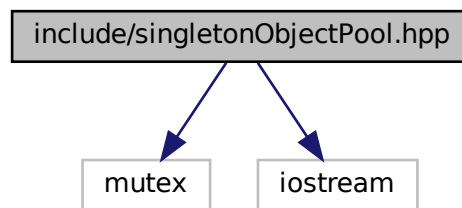
6.45 include/singletonObjectPool.hpp File Reference

Declare the [Tang::SingletonObjectPool](#) class.

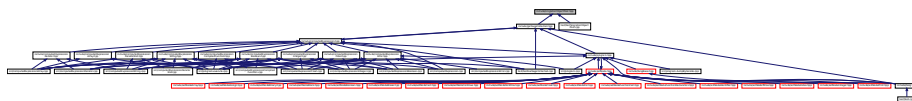
```
#include <mutex>
```

```
#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.

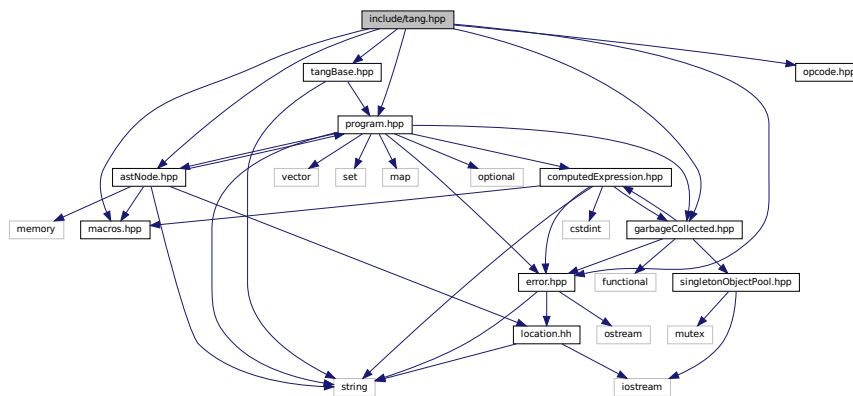
6.45.1 Detailed Description

Declare the [Tang::SingletonObjectPool](#) class.

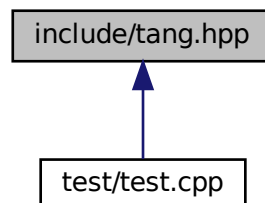
6.46 include/tang.hpp File Reference

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:



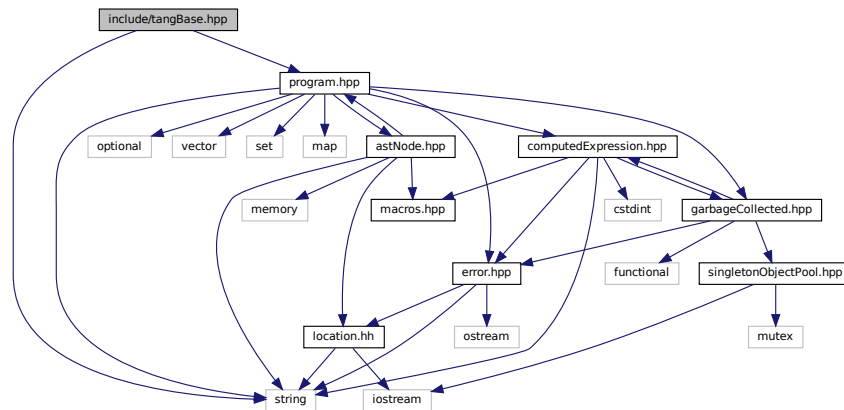
6.46.1 Detailed Description

Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

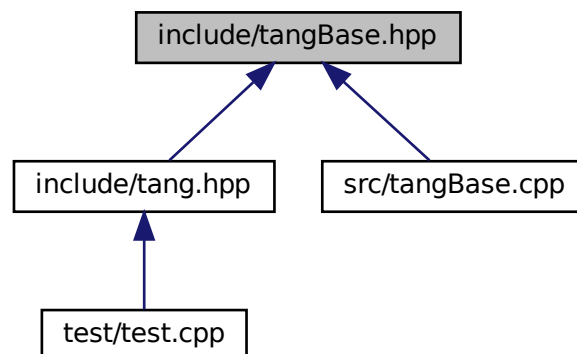
6.47 include/tangBase.hpp File Reference

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.

6.47.1 Detailed Description

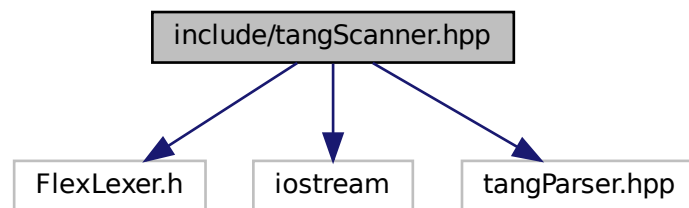
Declare the [Tang::TangBase](#) class used to interact with Tang.

6.48 include/tangScanner.hpp File Reference

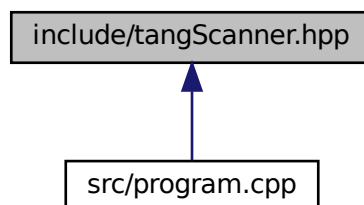
Declare the [Tang::TangScanner](#) used to tokenize a Tang script.

```
#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\(\)](#)

6.48.1 Detailed Description

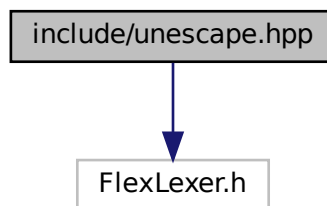
Declare the [Tang::TangScanner](#) used to tokenize a Tang script.

6.49 include/unescape.hpp File Reference

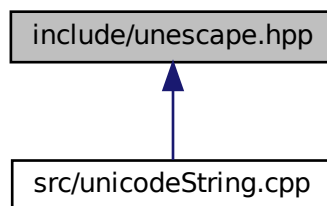
Declare the [Tang::Unescape](#) used to tokenize a Tang script.

```
#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\(\)](#)

6.49.1 Detailed Description

Declare the `Tang::Unescape` used to tokenize a Tang script.

6.50 include/unicodeString.hpp File Reference

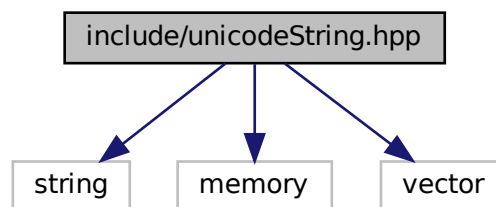
Contains the code to interface with the ICU library.

```
#include <string>
```

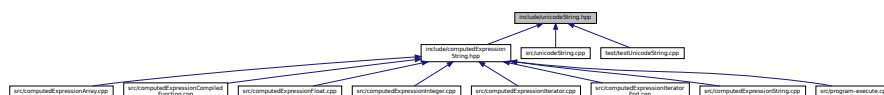
```
#include <memory>
```

```
#include <vector>
```

Include dependency graph for unicodeString.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class `Tang::UnicodeString`
Represents a UTF-8 encoded string that is Unicode-aware.

Functions

- std::string [Tang::unescape](#) (const std::string &str)
Return an "unescaped" version of the provided string, which, when interpreted by Tang, should result in a representation equivalent to the original source string.
- std::string [Tang::htmlEscape](#) (const std::string &str)
Return an "html escaped" version of the provided string.
- std::string [Tang::htmlEscapeAscii](#) (const std::string &str)
Return an Ascii-only, "html escaped" version of the provided string.

6.50.1 Detailed Description

Contains the code to interface with the ICU library.

6.50.2 Function Documentation

6.50.2.1 `htmlEscape()`

```
string Tang::htmlEscape (  
    const std::string & str )
```

Return an "html escaped" version of the provided string.

Only "critical" characters `<`, `>`, `&`, `"`, and ``` will be escaped. All other characters will be allowed through unaltered. The result is a UTF-8 encoded string that is safe for inclusion in an HTML template without disturbing the HTML structure.

Parameters

<code>str</code>	The string to be escaped.
------------------	---------------------------

Returns

An "escaped" version of the provided string.

Here is the call graph for this function:



6.50.2.2 `htmlEscapeAscii()`

```
string Tang::htmlEscapeAscii (  
    const std::string & str )
```

Return an Ascii-only, "html escaped" version of the provided string.

This function will convert all characters into an Ascii-only representation of the provided UTF-8 encoded string. Visible, standard Ascii characters will pass through unaltered, but all others will be replaced by their HTML escape sequence (if it exists), or the appropriate hexadecimal escape code.

Parameters

<i>str</i>	The string to be escaped.
------------	---------------------------

Returns

An "escaped" version of the provided string.

Here is the call graph for this function:

**6.50.2.3 unescape()**

```
string Tang::unescape (
    const std::string & str )
```

Return an "unescaped" version of the provided string, which, when interpreted by Tang, should result in a representation equivalent to the original source string.

Parameters

<i>str</i>	The string to be unescaped.
------------	-----------------------------

Returns

An "unescaped" version of the provided string.

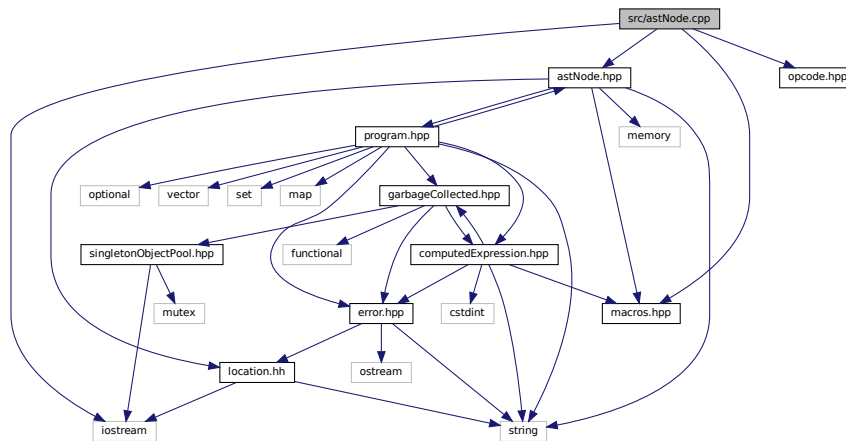
Here is the call graph for this function:



6.51 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:
```



6.51.1 Detailed Description

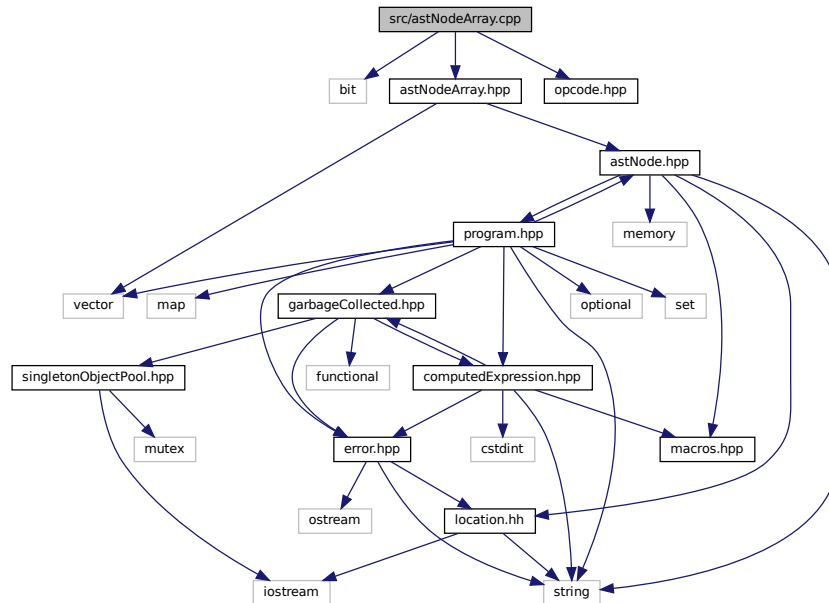
Define the [Tang::AstNode](#) class.

6.52 src/astNodeArray.cpp File Reference

Define the [Tang::AstNodeArray](#) class.

```
#include <bit>
#include "astNodeArray.hpp"
#include "opcode.hpp"
```

Include dependency graph for astNodeArray.cpp:



6.52.1 Detailed Description

Define the [Tang::AstNodeArray](#) class.

6.53 src/astNodeAssign.cpp File Reference

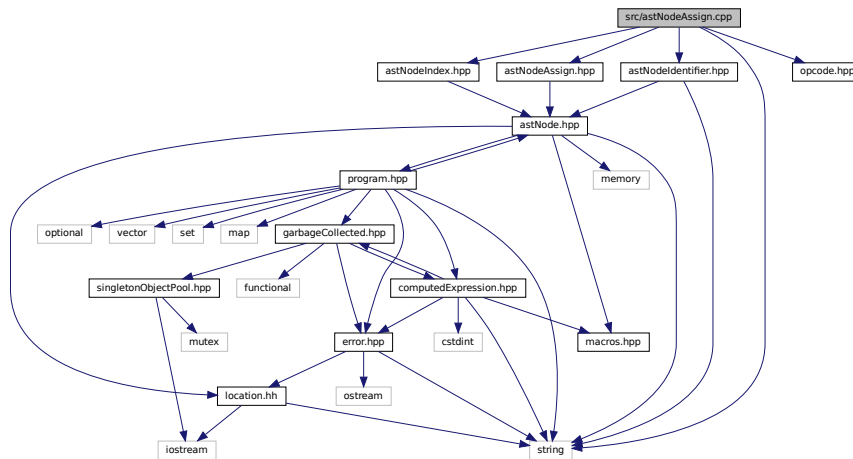
Define the [Tang::AstNodeAssign](#) class.

```

#include <string>
#include "astNodeAssign.hpp"
#include "astNodeIdentifier.hpp"
#include "astNodeIndex.hpp"
#include "opcode.hpp"

```

Include dependency graph for `astNodeAssign.cpp`:



6.53.1 Detailed Description

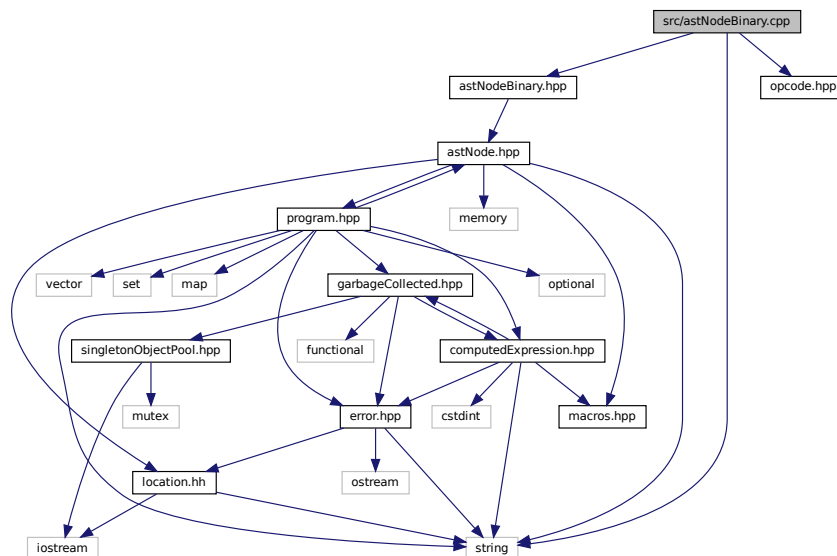
Define the [Tang::AstNodeAssign](#) class.

6.54 src/astNodeBinary.cpp File Reference

Define the [Tang::AstNodeBinary](#) class.

```
#include <string>
#include "astNodeBinary.hpp"
#include "opcode.hpp"
```

Include dependency graph for `astNodeBinary.cpp`:



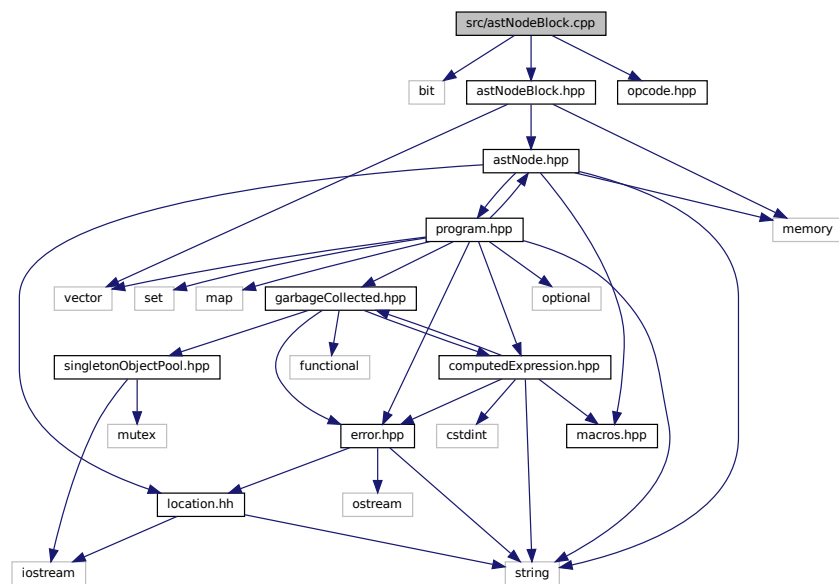
6.54.1 Detailed Description

Define the [Tang::AstNodeBinary](#) class.

6.55 src/astNodeBlock.cpp File Reference

Define the [Tang::AstNodeBlock](#) class.

```
#include <bit>
#include "astNodeBlock.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeBlock.cpp:
```



6.55.1 Detailed Description

Define the [Tang::AstNodeBlock](#) class.

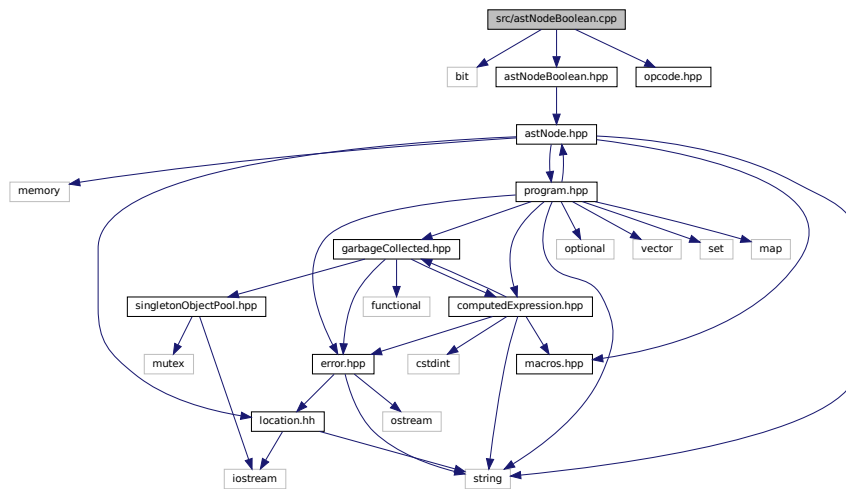
6.56 src/astNodeBoolean.cpp File Reference

Define the [Tang::AstNodeBoolean](#) class.

```
#include <bit>
#include "astNodeBoolean.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeBoolean.cpp:



6.56.1 Detailed Description

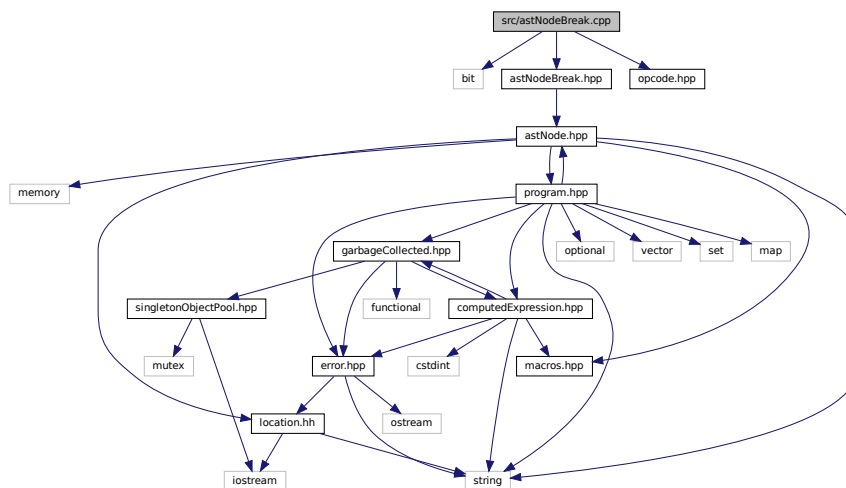
Define the [Tang::AstNodeBoolean](#) class.

6.57 src/astNodeBreak.cpp File Reference

Define the [Tang::AstNodeBreak](#) class.

```
#include <bit>
#include "astNodeBreak.hpp"
#include "opcode.hpp"
```

Include dependency graph for astNodeBreak.cpp:



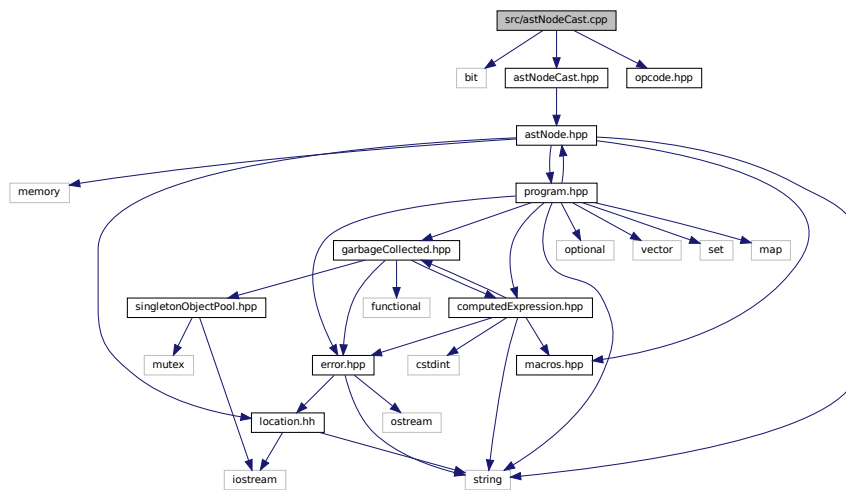
6.57.1 Detailed Description

Define the [Tang::AstNodeBreak](#) class.

6.58 src/astNodeCast.cpp File Reference

Define the [Tang::AstNodeCast](#) class.

```
#include <bit>
#include "astNodeCast.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeCast.cpp:
```



6.58.1 Detailed Description

Define the [Tang::AstNodeCast](#) class.

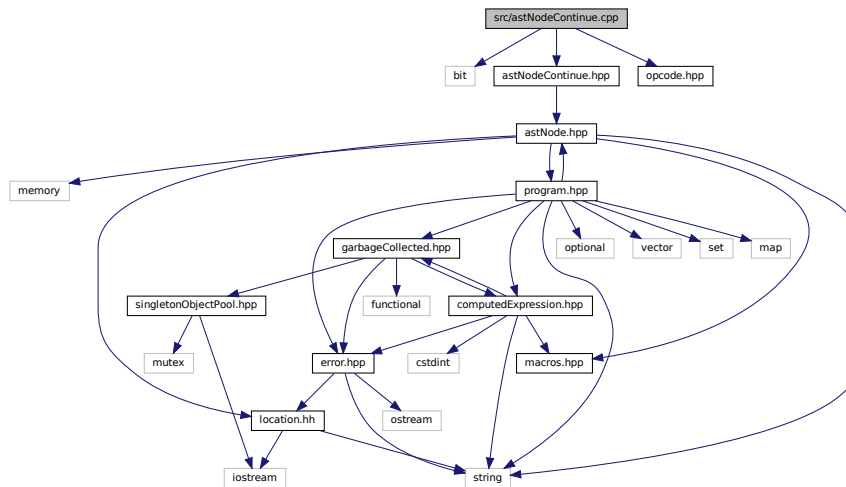
6.59 src/astNodeContinue.cpp File Reference

Define the [Tang::AstNodeContinue](#) class.

```
#include <bit>
#include "astNodeContinue.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for `astNodeContinue.cpp`:



6.59.1 Detailed Description

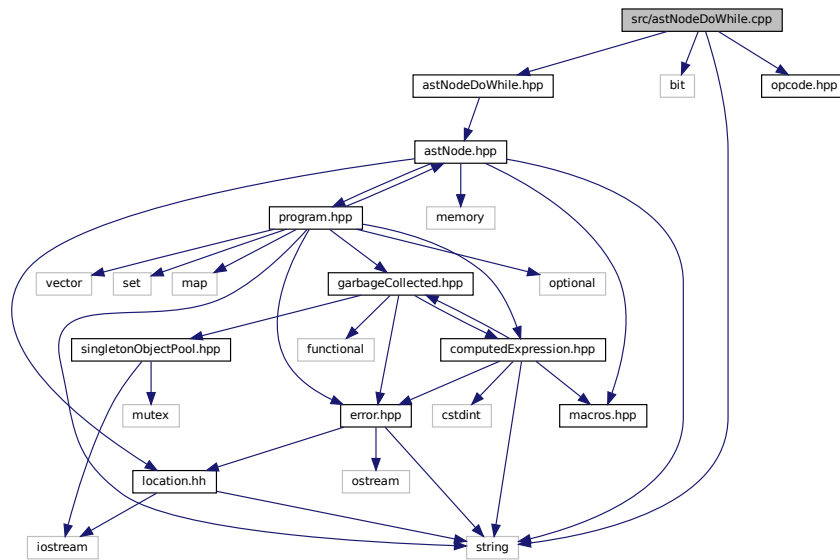
Define the [Tang::AstNodeContinue](#) class.

6.60 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:



6.60.1 Detailed Description

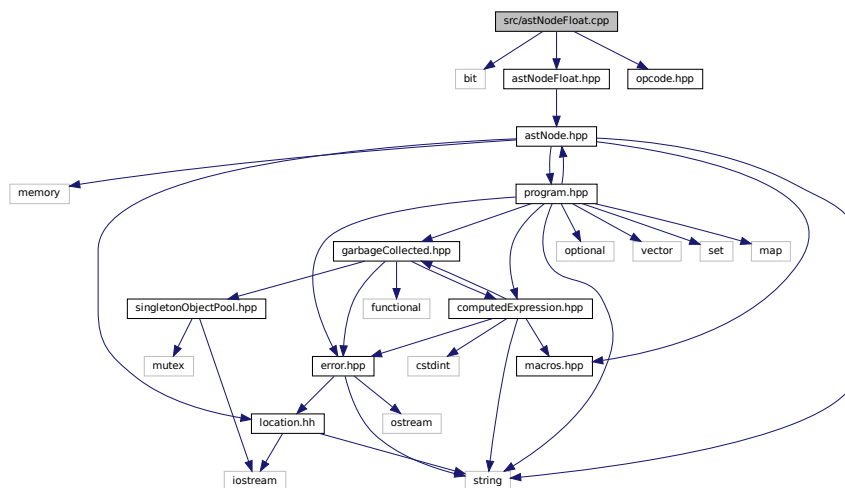
Define the [Tang::AstNodeDoWhile](#) class.

6.61 src/astNodeFloat.cpp File Reference

Define the [Tang::AstNodeFloat](#) class.

```
#include <bit>
#include "astNodeFloat.hpp"
#include "opcode.hpp"
```

Include dependency graph for astNodeFloat.cpp:



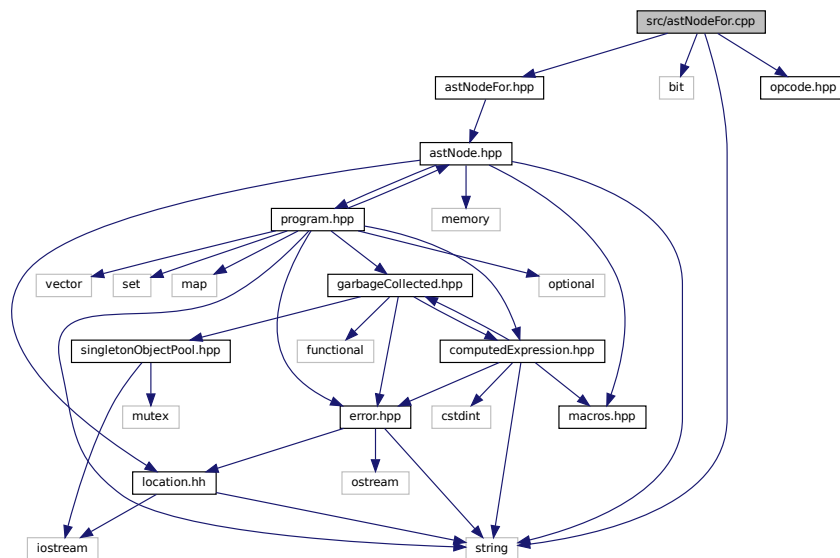
6.61.1 Detailed Description

Define the [Tang::AstNodeFloat](#) class.

6.62 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:
```



6.62.1 Detailed Description

Define the [Tang::AstNodeFor](#) class.

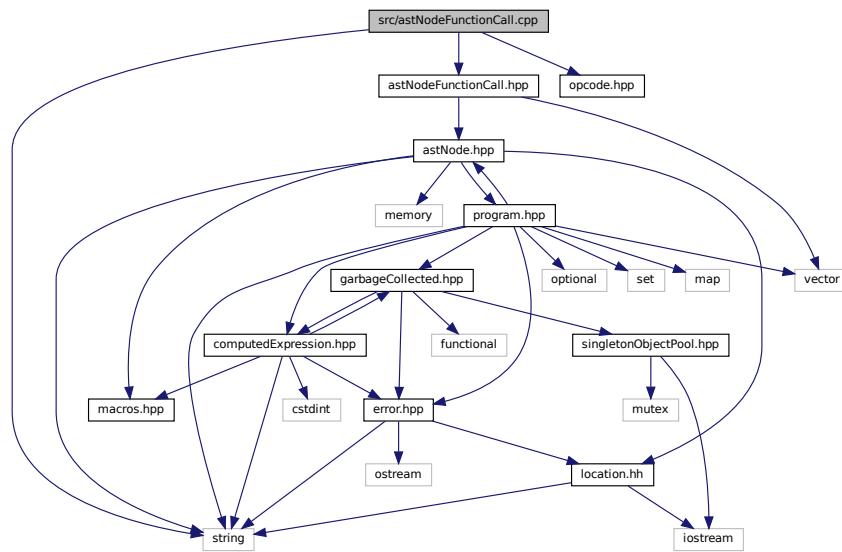
6.63 src/astNodeFunctionCall.cpp File Reference

Define the [Tang::AstNodeFunctionCall](#) class.

```
#include <string>
#include "astNodeFunctionCall.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeFunctionCall.cpp:



6.63.1 Detailed Description

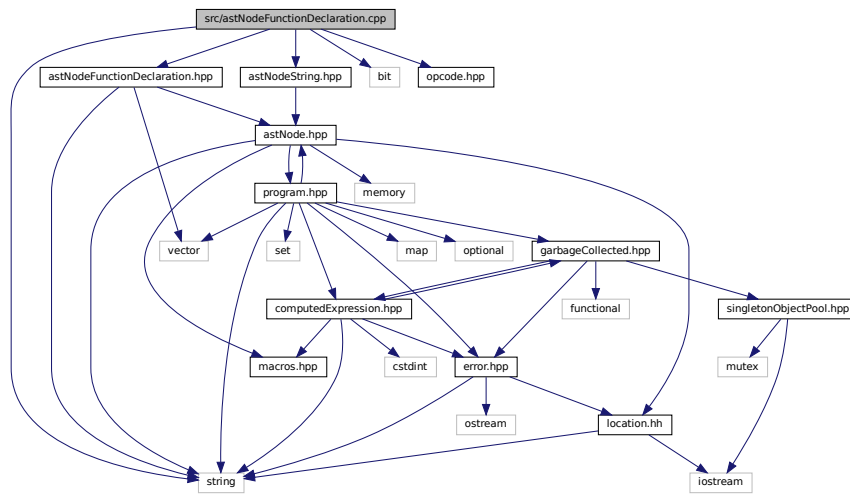
Define the [Tang::AstNodeFunctionCall](#) class.

6.64 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`:



6.64.1 Detailed Description

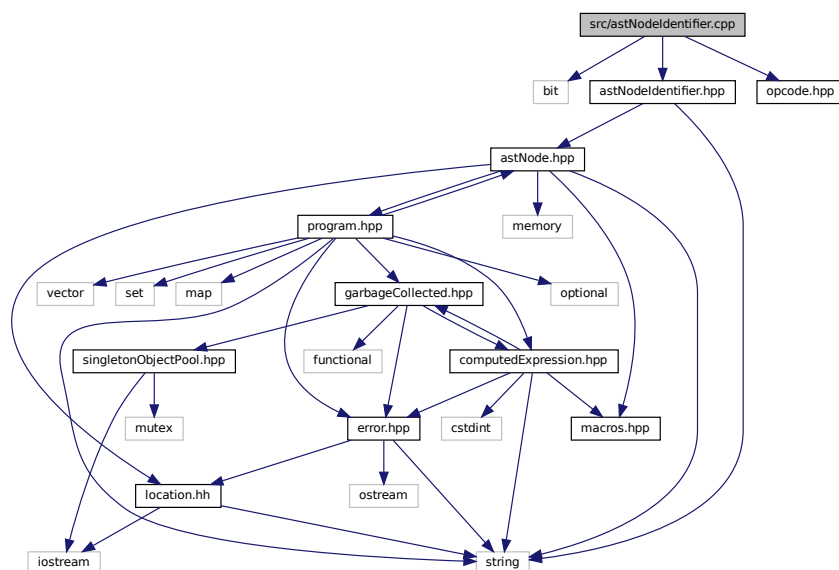
Define the [Tang::AstNodeFunctionDeclaration](#) class.

6.65 src/astNodeIdentifier.cpp File Reference

Define the [Tang::AstNodeIdentifier](#) class.

```
#include <bit>
#include "astNodeIdentifier.hpp"
#include "opcode.hpp"
```

Include dependency graph for `astNodeIdentifier.cpp`:



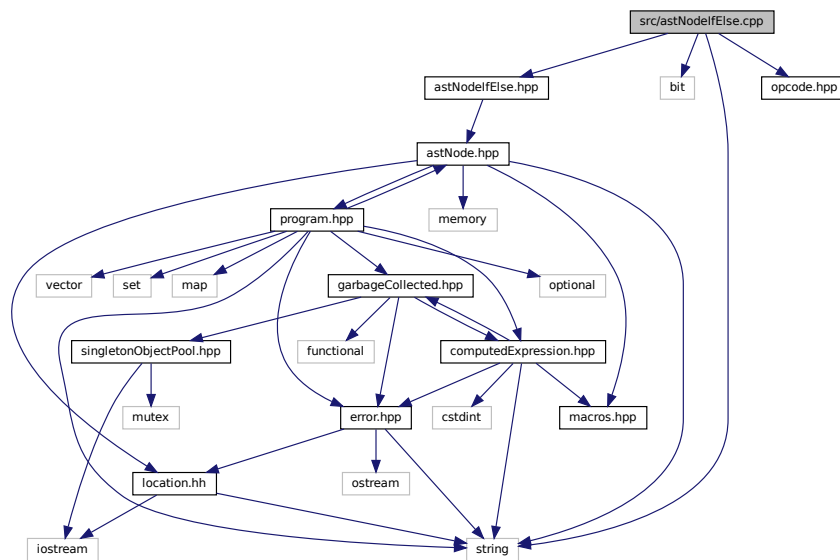
6.65.1 Detailed Description

Define the [Tang::AstNodeIdentifier](#) class.

6.66 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:
```



6.66.1 Detailed Description

Define the [Tang::AstNodeIfElse](#) class.

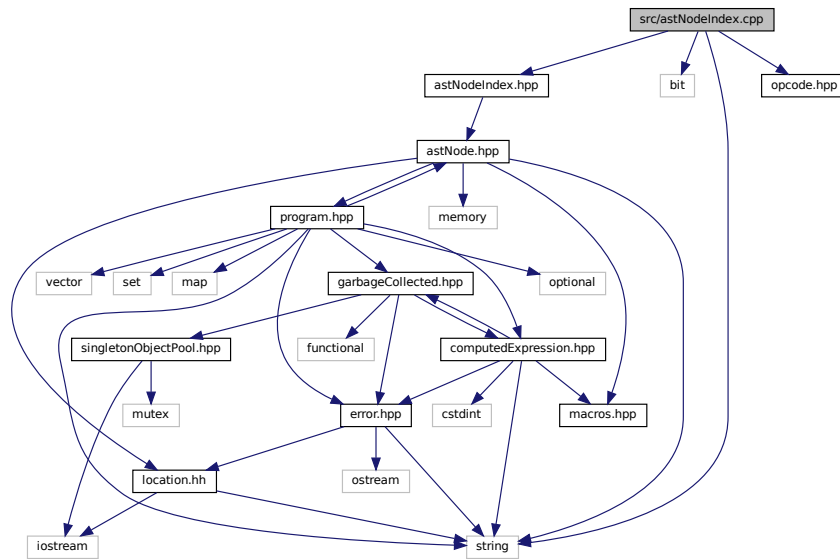
6.67 src/astNodeIndex.cpp File Reference

Define the [Tang::AstNodeIndex](#) class.

```
#include <string>
#include <bit>
#include "astNodeIndex.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeIndex.cpp:



6.67.1 Detailed Description

Define the [Tang::AstNodeIndex](#) class.

6.68 src/astNodeInteger.cpp File Reference

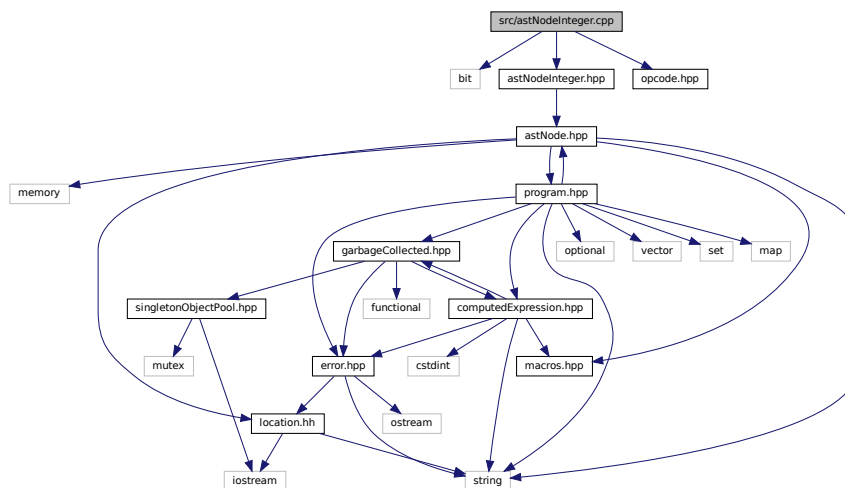
Define the [Tang::AstNodeInteger](#) class.

```
#include <bit>
```

```
#include "astNodeInteger.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeInteger.cpp:



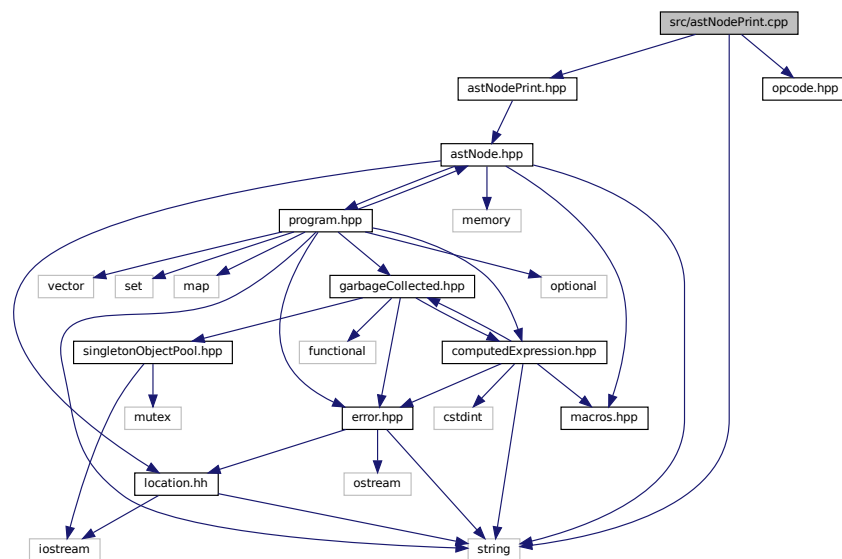
6.68.1 Detailed Description

Define the [Tang::AstNodeInteger](#) class.

6.69 src/astNodePrint.cpp File Reference

Define the [Tang::AstNodePrint](#) class.

```
#include <string>
#include "astNodePrint.hpp"
#include "opcode.hpp"
Include dependency graph for astNodePrint.cpp:
```



6.69.1 Detailed Description

Define the [Tang::AstNodePrint](#) class.

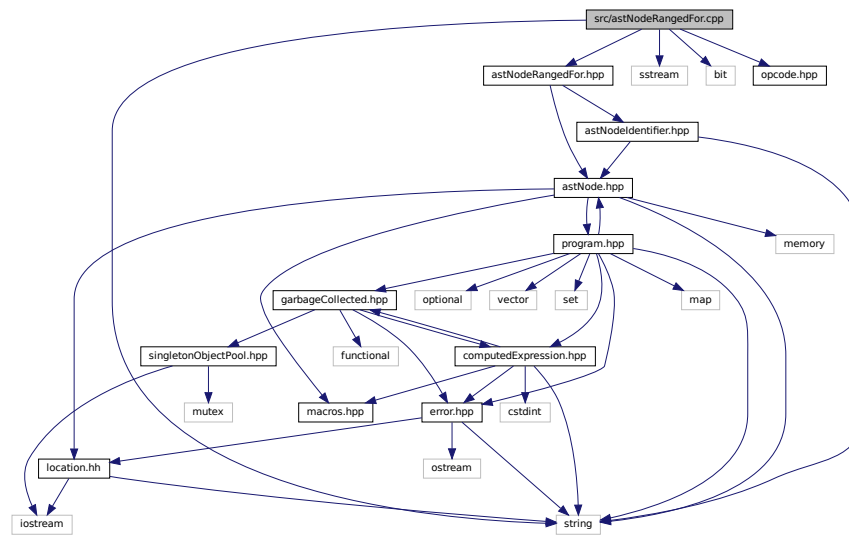
6.70 src/astNodeRangedFor.cpp File Reference

Define the [Tang::AstNodeRangedFor](#) class.

```
#include <string>
#include <sstream>
#include <bit>
#include "astNodeRangedFor.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for `astNodeRangedFor.cpp`:



6.70.1 Detailed Description

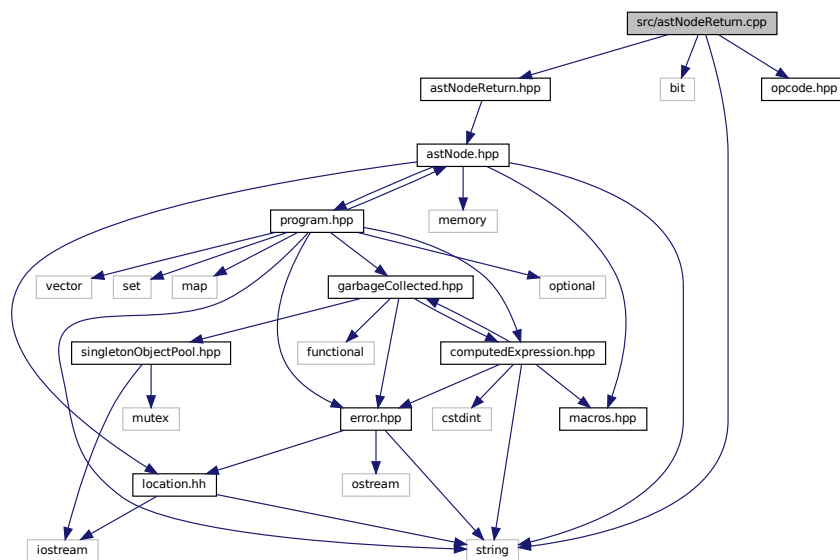
Define the [Tang::AstNodeRangedFor](#) class.

6.71 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:



6.71.1 Detailed Description

Define the [Tang::AstNodeReturn](#) class.

6.72 src/astNodeSlice.cpp File Reference

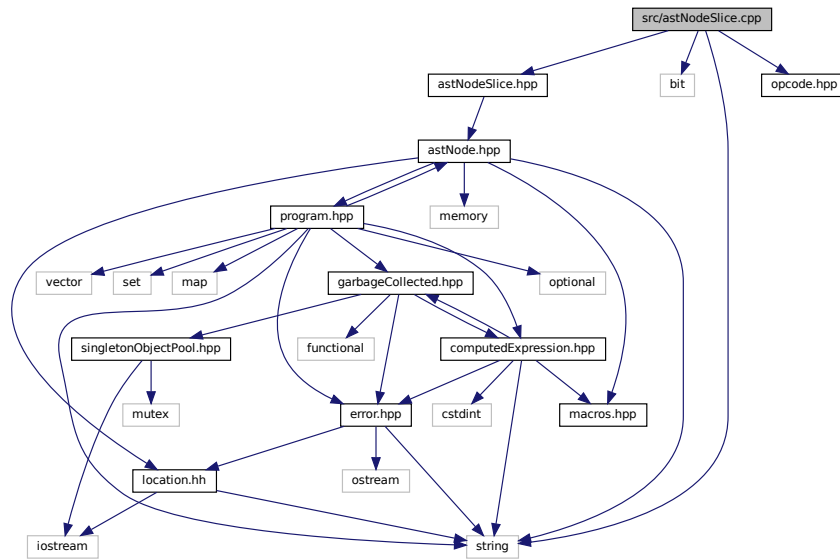
Define the [Tang::AstNodeSlice](#) class.

```

#include <string>
#include <bit>
#include "astNodeSlice.hpp"
#include "opcode.hpp"

```

Include dependency graph for astNodeSlice.cpp:



6.72.1 Detailed Description

Define the [Tang::AstNodeSlice](#) class.

6.73 src/astNodeString.cpp File Reference

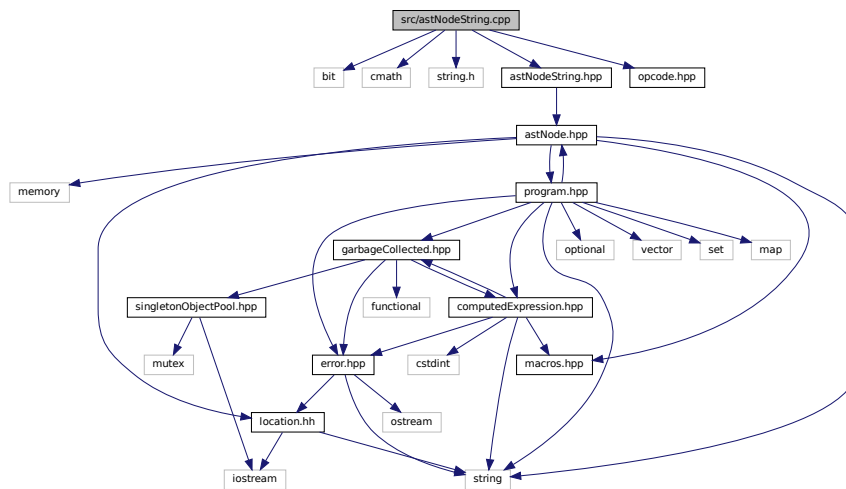
Define the [Tang::AstNodeString](#) class.

```

#include <bit>
#include <cmath>
#include <string.h>
#include "astNodeString.hpp"
#include "opcode.hpp"

```

Include dependency graph for astNodeString.cpp:



6.73.1 Detailed Description

Define the [Tang::AstNodeString](#) class.

6.74 src/astNodeTernary.cpp File Reference

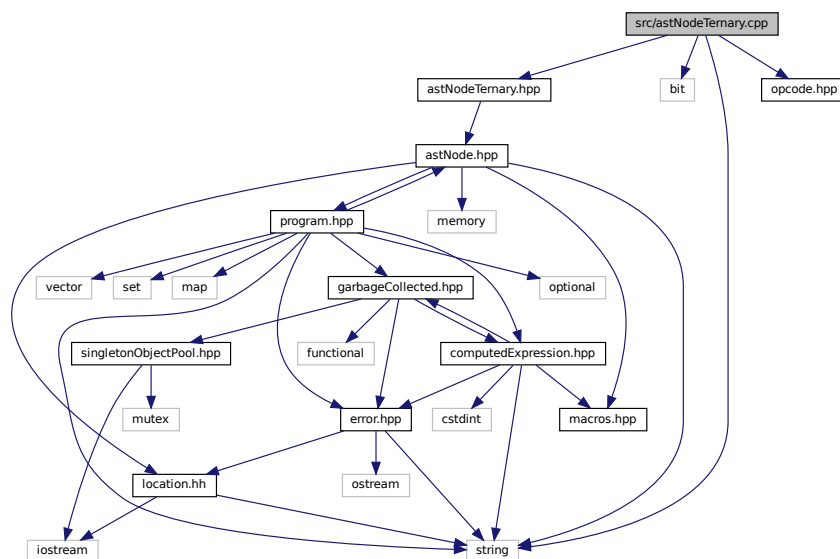
Define the [Tang::AstNodeTernary](#) class.

```

#include <string>
#include <bit>
#include "astNodeTernary.hpp"
#include "opcode.hpp"

```

Include dependency graph for astNodeTernary.cpp:



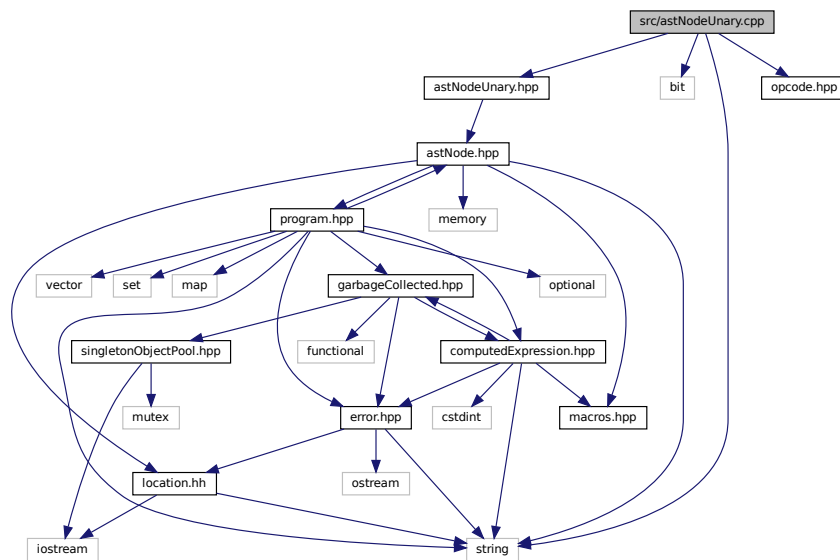
6.74.1 Detailed Description

Define the [Tang::AstNodeUnary](#) class.

6.75 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:
```



6.75.1 Detailed Description

Define the [Tang::AstNodeUnary](#) class.

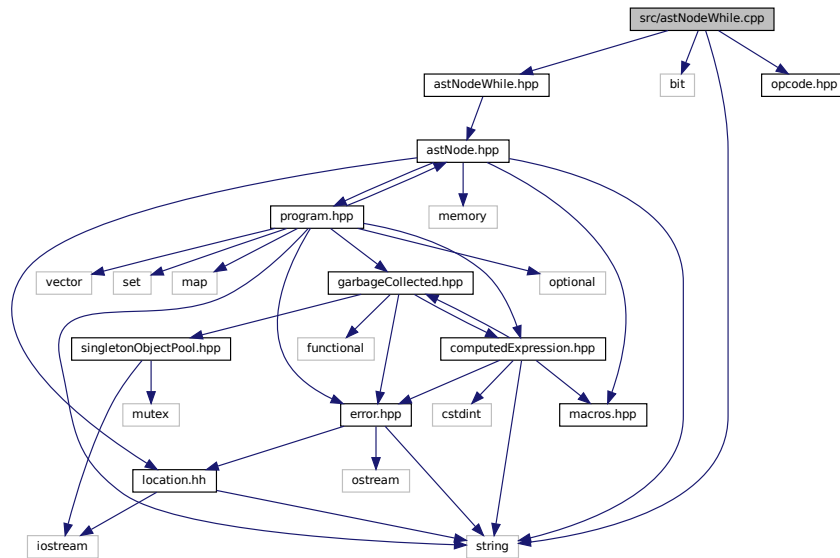
6.76 src/astNodeWhile.cpp File Reference

Define the [Tang::AstNodeWhile](#) class.

```
#include <string>
#include <bit>
#include "astNodeWhile.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeWhile.cpp:



6.76.1 Detailed Description

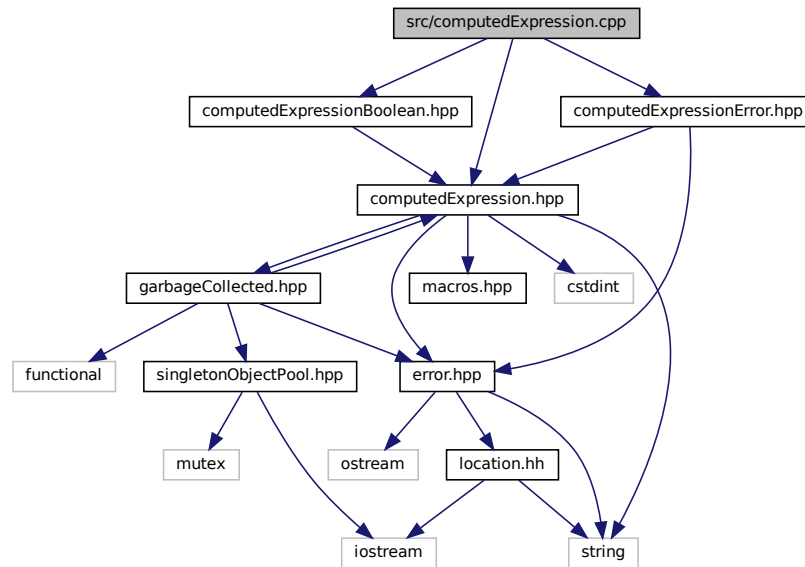
Define the [Tang::AstNodeWhile](#) class.

6.77 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`:



6.77.1 Detailed Description

Define the [Tang::ComputedExpression](#) class.

6.78 src/computedExpressionArray.cpp File Reference

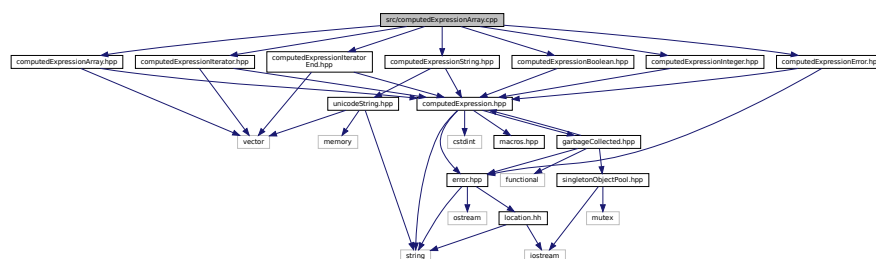
Define the [Tang::ComputedExpressionArray](#) class.

```

#include "computedExpressionArray.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionIterator.hpp"
#include "computedExpressionIteratorEnd.hpp"
#include "computedExpressionError.hpp"

```

Include dependency graph for `computedExpressionArray.cpp`:



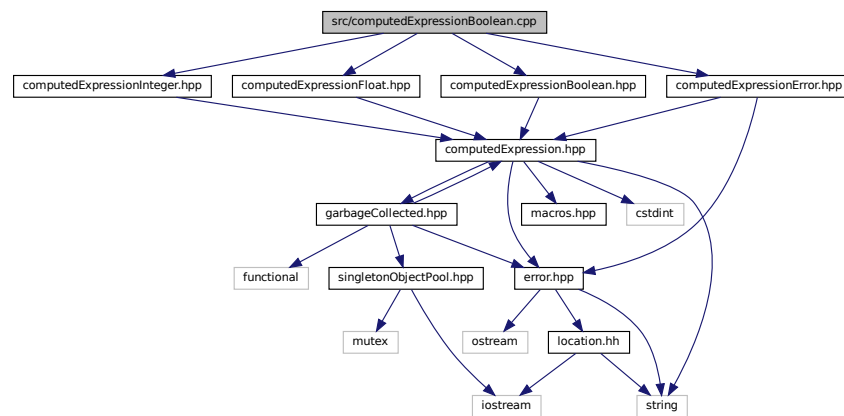
6.78.1 Detailed Description

Define the [Tang::ComputedExpressionArray](#) class.

6.79 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:
```



6.79.1 Detailed Description

Define the [Tang::ComputedExpressionBoolean](#) class.

6.80 src/computedExpressionCompiledFunction.cpp File Reference

Define the [Tang::ComputedExpressionCompiledFunction](#) class.

```
#include "computedExpressionCompiledFunction.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
```

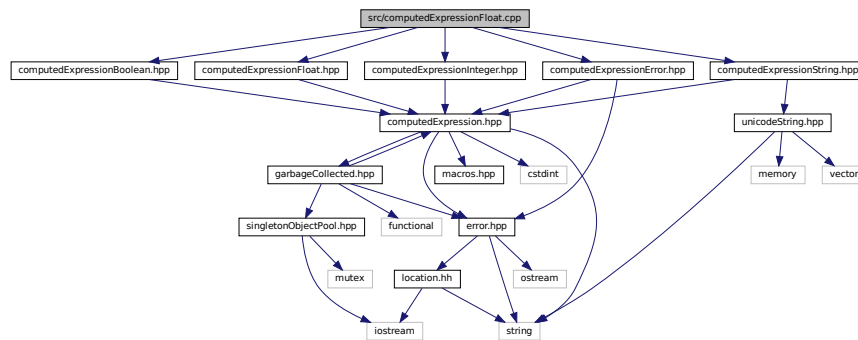

6.81.1 Detailed Description

Define the [Tang::ComputedExpressionError](#) class.

6.82 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:
```



6.82.1 Detailed Description

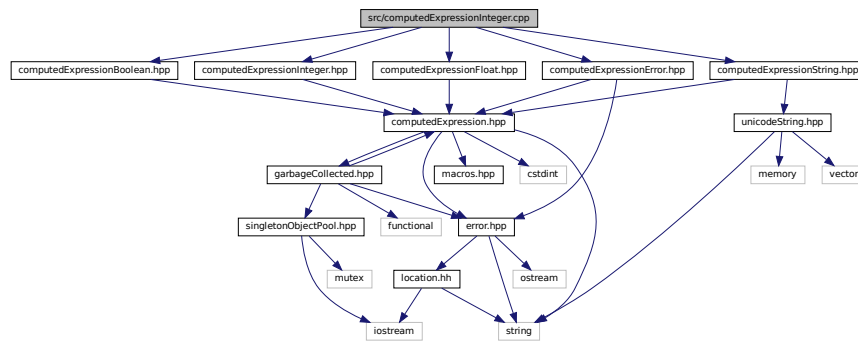
Define the [Tang::ComputedExpressionFloat](#) class.

6.83 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:
```



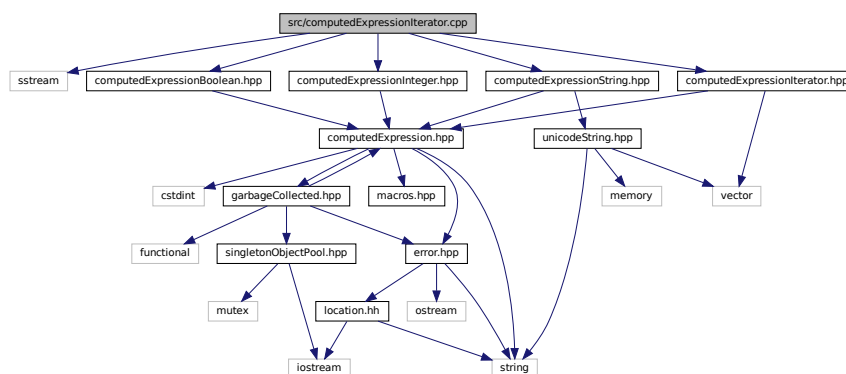
6.83.1 Detailed Description

Define the [Tang::ComputedExpressionInteger](#) class.

6.84 src/computedExpressionIterator.cpp File Reference

Define the [Tang::ComputedExpressionIterator](#) class.

```
#include <sstream>
#include "computedExpressionIterator.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
Include dependency graph for computedExpressionIterator.cpp:
```



6.84.1 Detailed Description

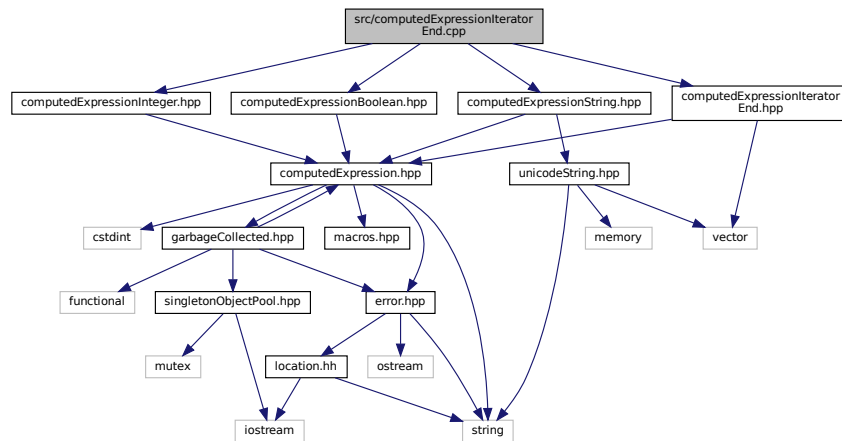
Define the [Tang::ComputedExpressionIterator](#) class.

6.85 src/computedExpressionIteratorEnd.cpp File Reference

Define the [Tang::ComputedExpressionIteratorEnd](#) class.

```
#include "computedExpressionIteratorEnd.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
```

Include dependency graph for computedExpressionIteratorEnd.cpp:



6.85.1 Detailed Description

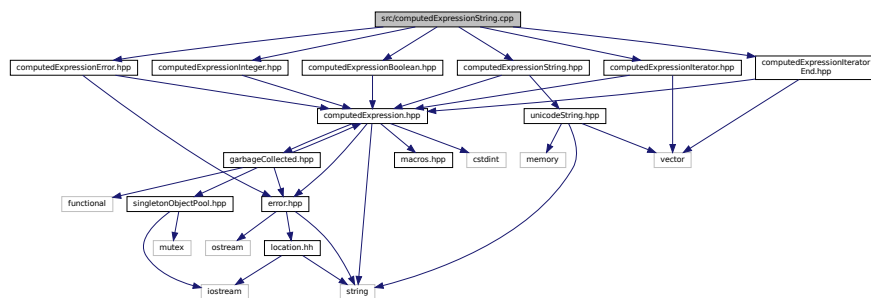
Define the [Tang::ComputedExpressionIteratorEnd](#) class.

6.86 src/computedExpressionString.cpp File Reference

Define the [Tang::ComputedExpressionString](#) class.

```
#include "computedExpressionString.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionError.hpp"
#include "computedExpressionInteger.hpp"
#include "computedExpressionIterator.hpp"
#include "computedExpressionIteratorEnd.hpp"
```

Include dependency graph for computedExpressionString.cpp:



6.86.1 Detailed Description

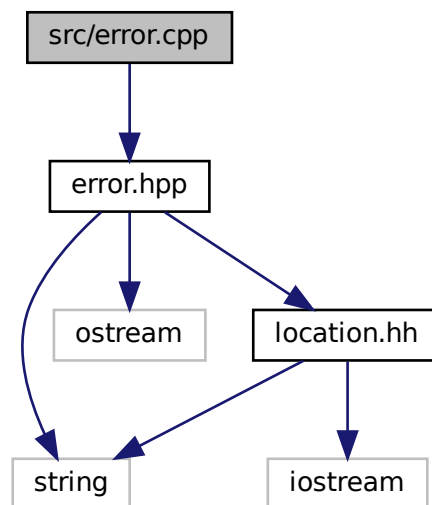
Define the [Tang::ComputedExpressionString](#) class.

6.87 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)`

6.87.1 Detailed Description

Define the [Tang::Error](#) class.

6.87.2 Function Documentation

6.87.2.1 `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

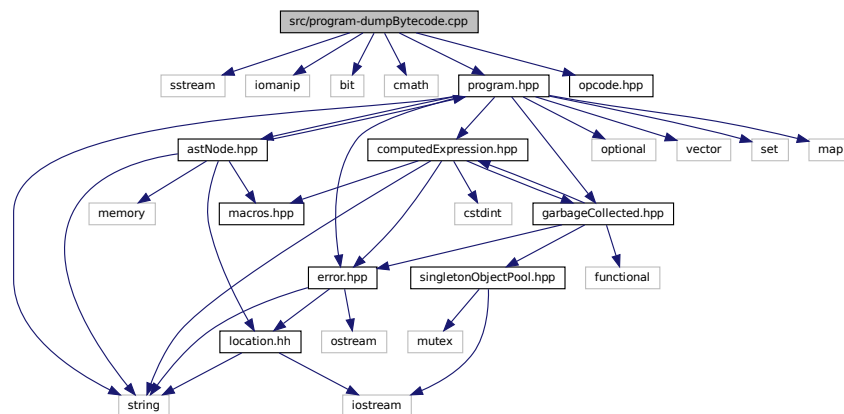
The output stream.

6.88 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.

6.88.1 Detailed Description

Define the [Tang::Program::dumpBytecode](#) method.

6.88.2 Macro Definition Documentation

6.88.2.1 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

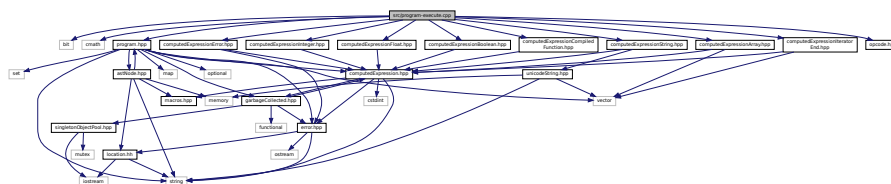
x	The number of additional vector entries that should exist.
---	--

6.89 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 "computedExpressionIteratorEnd.hpp"
```

Include dependency graph for program-execute.cpp:



Macros

- `#define EXECUTEPROGRAMCHECK(x)`
Verify the size of the Bytecode vector so that it may be safely accessed.
- `#define STACKCHECK(x)`
Verify the size of the stack vector so that it may be safely accessed.

6.89.1 Detailed Description

Define the [Tang::Program::execute](#) method.

6.89.2 Macro Definition Documentation

6.89.2.1 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.
---	--

6.89.2.2 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

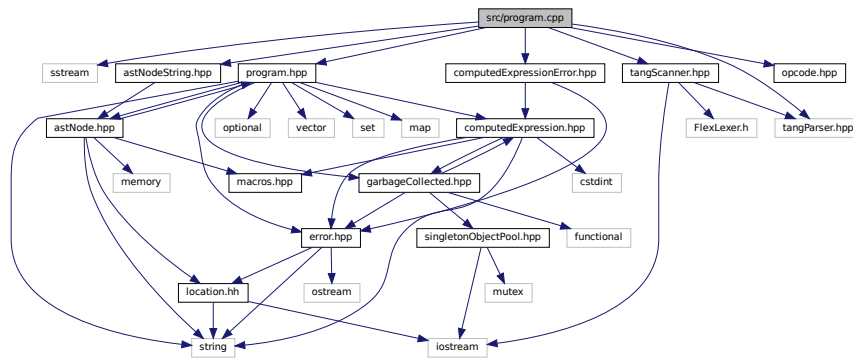
x	The number of entries that should exist in the stack.
---	---

6.90 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:



6.90.1 Detailed Description

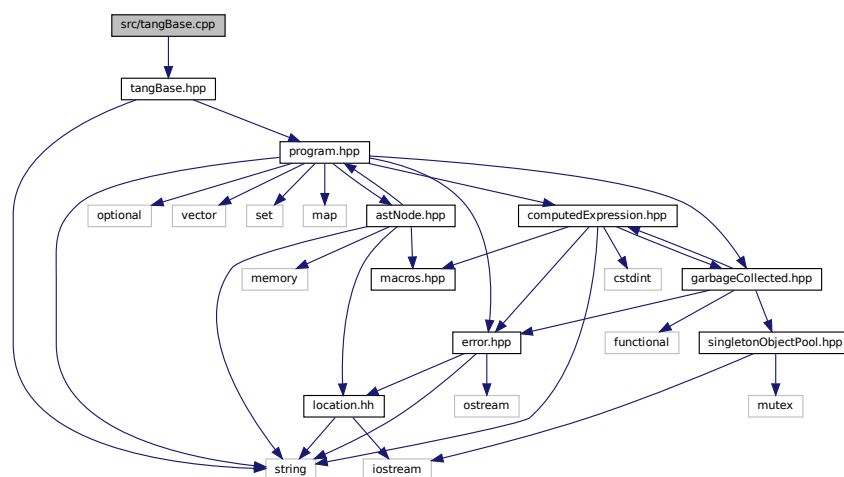
Define the [Tang::Program](#) class.

6.91 src/tangBase.cpp File Reference

Define the [Tang::TangBase](#) class.

```
#include "tangBase.hpp"
```

Include dependency graph for tangBase.cpp:



6.91.1 Detailed Description

Define the [Tang::TangBase](#) class.

6.92 src/unicodeString.cpp File Reference

Contains the function declarations for the [Tang::UnicodeString](#) class and the interface to ICU.

```
#include <cassert>
#include <vector>
#include <memory>
#include <algorithm>
#include <sstream>
#include <unicode/uconfig.h>
#include <unicode/ustring.h>
#include <unicode/brkiter.h>
#include "unicodeString.hpp"
#include "unescape.hpp"
#include "htmlEscape.hpp"
#include "htmlEscapeAscii.hpp"
Include dependency graph for unicodeString.cpp:
```



6.92.1 Detailed Description

Contains the function declarations for the [Tang::UnicodeString](#) class and the interface to ICU.

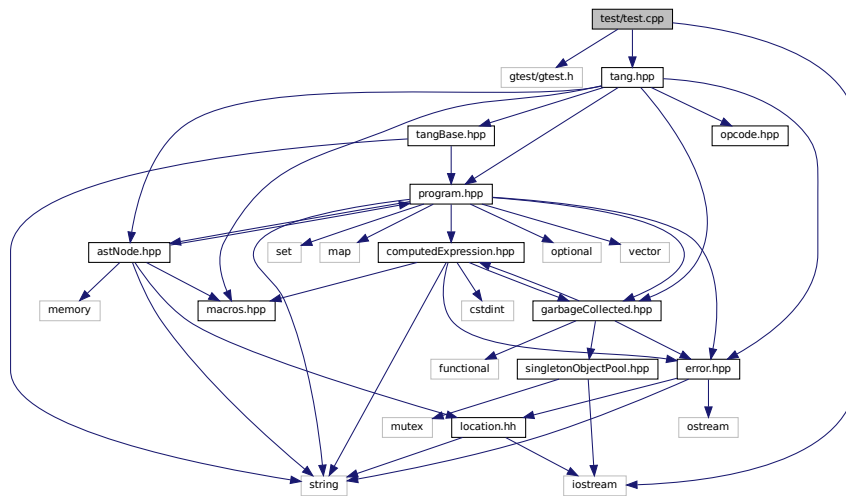
6.93 test/test.cpp File Reference

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** (ControlFlow, RangedFor)
- **TEST** (Print, Default)
- **TEST** (Print, Array)

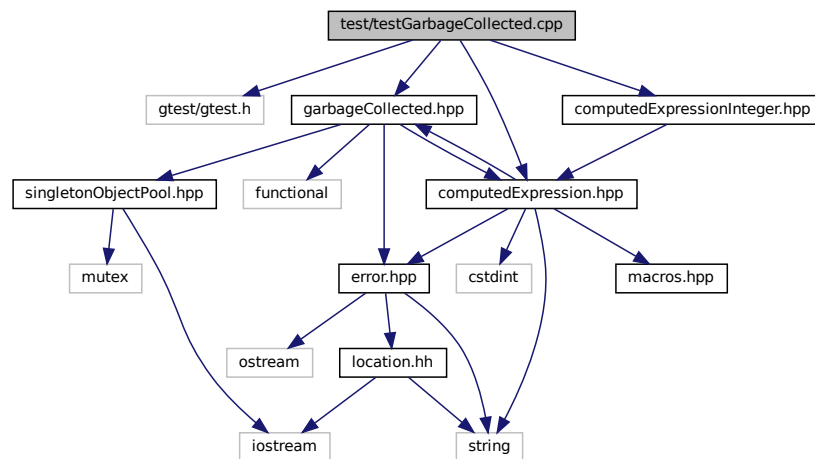
6.93.1 Detailed Description

Test the general language behaviors.

6.94 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, ObjectsIsRecycled)
- **TEST** (Recycle, ObjectsIsNotRecycled)
- **int main** (int argc, char **argv)

6.94.1 Detailed Description

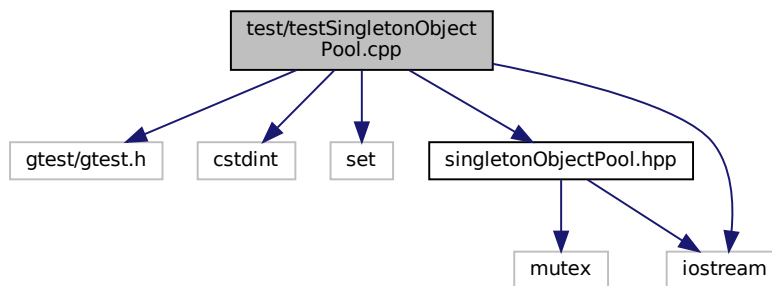
Test the generic behavior of the [Tang::GarbageCollected](#) class.

6.95 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)

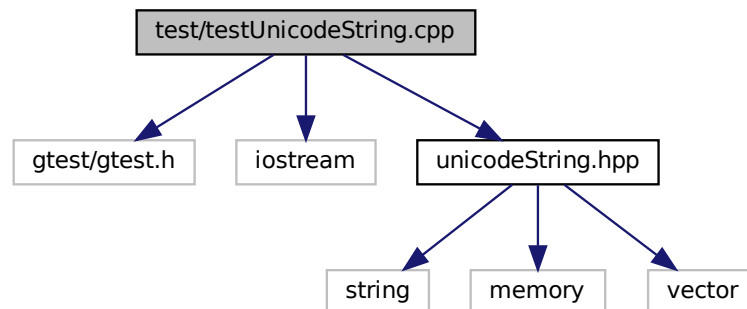
6.95.1 Detailed Description

Test the generic behavior of the [Tang::SingletonObjectPool](#) class.

6.96 test/testUnicodeString.cpp File Reference

Contains tests for the [Tang::UnicodeString](#) class.

```
#include <gtest/gtest.h>
#include <iostream>
#include "unicodeString.hpp"
Include dependency graph for testUnicodeString.cpp:
```



Functions

- **TEST** (Core, [Unescape](#))
- **TEST** (Core, [HtmlEscape](#))
- **TEST** (Core, [HtmlEscapeAscii](#))
- **TEST** ([UnicodeString](#), SubString)
- `int main` (int argc, char **argv)

6.96.1 Detailed Description

Contains tests for the [Tang::UnicodeString](#) class.

Index

- __add
 - Tang::ComputedExpression, [131](#)
 - Tang::ComputedExpressionArray, [144](#)
 - Tang::ComputedExpressionBoolean, [157](#)
 - Tang::ComputedExpressionCompiledFunction, [169](#)
 - Tang::ComputedExpressionError, [183](#)
 - Tang::ComputedExpressionFloat, [195](#)
 - Tang::ComputedExpressionInteger, [210](#)
 - Tang::ComputedExpressionIterator, [226](#)
 - Tang::ComputedExpressionIteratorEnd, [239](#)
 - Tang::ComputedExpressionString, [252](#)
- __asCode
 - Tang::ComputedExpression, [131](#)
 - Tang::ComputedExpressionArray, [144](#)
 - Tang::ComputedExpressionBoolean, [157](#)
 - Tang::ComputedExpressionCompiledFunction, [170](#)
 - Tang::ComputedExpressionError, [183](#)
 - Tang::ComputedExpressionFloat, [197](#)
 - Tang::ComputedExpressionInteger, [212](#)
 - Tang::ComputedExpressionIterator, [226](#)
 - Tang::ComputedExpressionIteratorEnd, [239](#)
 - Tang::ComputedExpressionString, [252](#)
- __assign_index
 - Tang::ComputedExpression, [131](#)
 - Tang::ComputedExpressionArray, [144](#)
 - Tang::ComputedExpressionBoolean, [158](#)
 - Tang::ComputedExpressionCompiledFunction, [170](#)
 - Tang::ComputedExpressionError, [184](#)
 - Tang::ComputedExpressionFloat, [197](#)
 - Tang::ComputedExpressionInteger, [212](#)
 - Tang::ComputedExpressionIterator, [226](#)
 - Tang::ComputedExpressionIteratorEnd, [239](#)
 - Tang::ComputedExpressionString, [253](#)
- __boolean
 - Tang::ComputedExpression, [132](#)
 - Tang::ComputedExpressionArray, [145](#)
 - Tang::ComputedExpressionBoolean, [158](#)
 - Tang::ComputedExpressionCompiledFunction, [170](#)
 - Tang::ComputedExpressionError, [184](#)
 - Tang::ComputedExpressionFloat, [198](#)
 - Tang::ComputedExpressionInteger, [213](#)
 - Tang::ComputedExpressionIterator, [227](#)
 - Tang::ComputedExpressionIteratorEnd, [240](#)
 - Tang::ComputedExpressionString, [253](#)
- __divide
 - Tang::ComputedExpression, [132](#)
 - Tang::ComputedExpressionArray, [145](#)
 - Tang::ComputedExpressionBoolean, [158](#)
 - Tang::ComputedExpressionCompiledFunction, [171](#)
 - Tang::ComputedExpressionError, [184](#)
 - Tang::ComputedExpressionFloat, [198](#)
 - Tang::ComputedExpressionInteger, [213](#)
 - Tang::ComputedExpressionIterator, [227](#)
 - Tang::ComputedExpressionIteratorEnd, [240](#)
 - Tang::ComputedExpressionString, [254](#)
- __equal
 - Tang::ComputedExpression, [132](#)
 - Tang::ComputedExpressionArray, [146](#)
 - Tang::ComputedExpressionBoolean, [159](#)
 - Tang::ComputedExpressionCompiledFunction, [171](#)
 - Tang::ComputedExpressionError, [185](#)
 - Tang::ComputedExpressionFloat, [198](#)
 - Tang::ComputedExpressionInteger, [213](#)
 - Tang::ComputedExpressionIterator, [228](#)
 - Tang::ComputedExpressionIteratorEnd, [241](#)
 - Tang::ComputedExpressionString, [254](#)
- __float
 - Tang::ComputedExpression, [133](#)
 - Tang::ComputedExpressionArray, [146](#)
 - Tang::ComputedExpressionBoolean, [159](#)
 - Tang::ComputedExpressionCompiledFunction, [172](#)
 - Tang::ComputedExpressionError, [185](#)
 - Tang::ComputedExpressionFloat, [199](#)
 - Tang::ComputedExpressionInteger, [214](#)
 - Tang::ComputedExpressionIterator, [228](#)
 - Tang::ComputedExpressionIteratorEnd, [241](#)
 - Tang::ComputedExpressionString, [255](#)
- __getIterator
 - Tang::ComputedExpression, [133](#)
 - Tang::ComputedExpressionArray, [146](#)
 - Tang::ComputedExpressionBoolean, [159](#)
 - Tang::ComputedExpressionCompiledFunction, [172](#)
 - Tang::ComputedExpressionError, [185](#)
 - Tang::ComputedExpressionFloat, [199](#)
 - Tang::ComputedExpressionInteger, [214](#)
 - Tang::ComputedExpressionIterator, [228](#)
 - Tang::ComputedExpressionIteratorEnd, [241](#)
 - Tang::ComputedExpressionString, [255](#)
- __index
 - Tang::ComputedExpression, [133](#)
 - Tang::ComputedExpressionArray, [147](#)
 - Tang::ComputedExpressionBoolean, [160](#)
 - Tang::ComputedExpressionCompiledFunction, [172](#)
 - Tang::ComputedExpressionError, [186](#)
 - Tang::ComputedExpressionFloat, [200](#)
 - Tang::ComputedExpressionInteger, [215](#)
 - Tang::ComputedExpressionIterator, [229](#)
 - Tang::ComputedExpressionIteratorEnd, [242](#)

- Tang::ComputedExpressionString, 255
- __integer
 - Tang::ComputedExpression, 134
 - Tang::ComputedExpressionArray, 147
 - Tang::ComputedExpressionBoolean, 160
 - Tang::ComputedExpressionCompiledFunction, 173
 - Tang::ComputedExpressionError, 186
 - Tang::ComputedExpressionFloat, 200
 - Tang::ComputedExpressionInteger, 215
 - Tang::ComputedExpressionIterator, 229
 - Tang::ComputedExpressionIteratorEnd, 242
 - Tang::ComputedExpressionString, 256
- __iteratorNext
 - Tang::ComputedExpression, 134
 - Tang::ComputedExpressionArray, 147
 - Tang::ComputedExpressionBoolean, 160
 - Tang::ComputedExpressionCompiledFunction, 173
 - Tang::ComputedExpressionError, 186
 - Tang::ComputedExpressionFloat, 200
 - Tang::ComputedExpressionInteger, 215
 - Tang::ComputedExpressionIterator, 229
 - Tang::ComputedExpressionIteratorEnd, 242
 - Tang::ComputedExpressionString, 256
- __lessThan
 - Tang::ComputedExpression, 134
 - Tang::ComputedExpressionArray, 148
 - Tang::ComputedExpressionBoolean, 161
 - Tang::ComputedExpressionCompiledFunction, 173
 - Tang::ComputedExpressionError, 187
 - Tang::ComputedExpressionFloat, 201
 - Tang::ComputedExpressionInteger, 216
 - Tang::ComputedExpressionIterator, 230
 - Tang::ComputedExpressionIteratorEnd, 243
 - Tang::ComputedExpressionString, 257
- __modulo
 - Tang::ComputedExpression, 135
 - Tang::ComputedExpressionArray, 148
 - Tang::ComputedExpressionBoolean, 161
 - Tang::ComputedExpressionCompiledFunction, 174
 - Tang::ComputedExpressionError, 187
 - Tang::ComputedExpressionFloat, 201
 - Tang::ComputedExpressionInteger, 216
 - Tang::ComputedExpressionIterator, 230
 - Tang::ComputedExpressionIteratorEnd, 243
 - Tang::ComputedExpressionString, 257
- __multiply
 - Tang::ComputedExpression, 135
 - Tang::ComputedExpressionArray, 149
 - Tang::ComputedExpressionBoolean, 161
 - Tang::ComputedExpressionCompiledFunction, 174
 - Tang::ComputedExpressionError, 187
 - Tang::ComputedExpressionFloat, 202
 - Tang::ComputedExpressionInteger, 217
 - Tang::ComputedExpressionIterator, 231
 - Tang::ComputedExpressionIteratorEnd, 243
 - Tang::ComputedExpressionString, 258
- __negative
 - Tang::ComputedExpression, 135
- Tang::ComputedExpressionArray, 149
- Tang::ComputedExpressionBoolean, 162
- Tang::ComputedExpressionCompiledFunction, 174
- Tang::ComputedExpressionError, 188
- Tang::ComputedExpressionFloat, 202
- Tang::ComputedExpressionInteger, 217
- Tang::ComputedExpressionIterator, 231
- Tang::ComputedExpressionIteratorEnd, 244
- Tang::ComputedExpressionString, 258
- __not
 - Tang::ComputedExpression, 136
 - Tang::ComputedExpressionArray, 149
 - Tang::ComputedExpressionBoolean, 162
 - Tang::ComputedExpressionCompiledFunction, 175
 - Tang::ComputedExpressionError, 188
 - Tang::ComputedExpressionFloat, 202
 - Tang::ComputedExpressionInteger, 218
 - Tang::ComputedExpressionIterator, 231
 - Tang::ComputedExpressionIteratorEnd, 244
 - Tang::ComputedExpressionString, 258
- __slice
 - Tang::ComputedExpression, 136
 - Tang::ComputedExpressionArray, 149
 - Tang::ComputedExpressionBoolean, 162
 - Tang::ComputedExpressionCompiledFunction, 175
 - Tang::ComputedExpressionError, 188
 - Tang::ComputedExpressionFloat, 203
 - Tang::ComputedExpressionInteger, 218
 - Tang::ComputedExpressionIterator, 231
 - Tang::ComputedExpressionIteratorEnd, 244
 - Tang::ComputedExpressionString, 259
- __string
 - Tang::ComputedExpression, 137
 - Tang::ComputedExpressionArray, 150
 - Tang::ComputedExpressionBoolean, 163
 - Tang::ComputedExpressionCompiledFunction, 176
 - Tang::ComputedExpressionError, 189
 - Tang::ComputedExpressionFloat, 203
 - Tang::ComputedExpressionInteger, 219
 - Tang::ComputedExpressionIterator, 232
 - Tang::ComputedExpressionIteratorEnd, 245
 - Tang::ComputedExpressionString, 260
- __subtract
 - Tang::ComputedExpression, 137
 - Tang::ComputedExpressionArray, 151
 - Tang::ComputedExpressionBoolean, 163
 - Tang::ComputedExpressionCompiledFunction, 176
 - Tang::ComputedExpressionError, 189
 - Tang::ComputedExpressionFloat, 204
 - Tang::ComputedExpressionInteger, 219
 - Tang::ComputedExpressionIterator, 232
 - Tang::ComputedExpressionIteratorEnd, 245
 - Tang::ComputedExpressionString, 260
- ~GarbageCollected
 - Tang::GarbageCollected, 269
- ADD
 - opcode.hpp, 358
- Add

- Tang::AstNodeBinary, [29](#)
- addBreak
 - Tang::Program, [294](#)
- addBytecode
 - Tang::Program, [294](#)
- addContinue
 - Tang::Program, [294](#)
- addIdentifier
 - Tang::Program, [295](#)
- addIdentifierAssigned
 - Tang::Program, [295](#)
- addString
 - Tang::Program, [295](#)
- And
 - Tang::AstNodeBinary, [29](#)
- ARRAY
 - opcode.hpp, [358](#)
- ASSIGNINDEX
 - opcode.hpp, [358](#)
- AstNode
 - Tang::AstNode, [16](#)
- AstNodeArray
 - Tang::AstNodeArray, [21](#)
- AstNodeAssign
 - Tang::AstNodeAssign, [25](#)
- AstNodeBinary
 - Tang::AstNodeBinary, [30](#)
- AstNodeBlock
 - Tang::AstNodeBlock, [34](#)
- AstNodeBoolean
 - Tang::AstNodeBoolean, [38](#)
- AstNodeBreak
 - Tang::AstNodeBreak, [42](#)
- AstNodeCast
 - Tang::AstNodeCast, [47](#)
- AstNodeContinue
 - Tang::AstNodeContinue, [50](#)
- AstNodeDoWhile
 - Tang::AstNodeDoWhile, [55](#)
- AstNodeFloat
 - Tang::AstNodeFloat, [59](#)
- AstNodeFor
 - Tang::AstNodeFor, [63](#)
- AstNodeFunctionCall
 - Tang::AstNodeFunctionCall, [67](#)
- AstNodeFunctionDeclaration
 - Tang::AstNodeFunctionDeclaration, [71](#)
- AstNodeIdentifier
 - Tang::AstNodeIdentifier, [75](#)
- AstNodeIfElse
 - Tang::AstNodeIfElse, [80](#), [81](#)
- AstNodeIndex
 - Tang::AstNodeIndex, [85](#)
- AstNodeInteger
 - Tang::AstNodeInteger, [90](#)
- AstNodePrint
 - Tang::AstNodePrint, [95](#)
- AstNodeRangedFor
 - Tang::AstNodeRangedFor, [98](#)
- AstNodeReturn
 - Tang::AstNodeReturn, [102](#)
- AstNodeSlice
 - Tang::AstNodeSlice, [107](#)
- AstNodeString
 - Tang::AstNodeString, [112](#)
- AstNodeTernary
 - Tang::AstNodeTernary, [117](#)
- AstNodeUnary
 - Tang::AstNodeUnary, [122](#)
- AstNodeWhile
 - Tang::AstNodeWhile, [126](#)
- BOOLEAN
 - opcode.hpp, [358](#)
- Boolean
 - Tang::AstNodeCast, [46](#)
- build/generated/location.hh, [315](#)
- bytesLength
 - Tang::UnicodeString, [311](#)
- CALLFUNC
 - opcode.hpp, [359](#)
- CASTBOOLEAN
 - opcode.hpp, [359](#)
- CASTFLOAT
 - opcode.hpp, [359](#)
- CASTINTEGER
 - opcode.hpp, [359](#)
- CodeType
 - Tang::Program, [293](#)
- compile
 - Tang::AstNode, [17](#)
 - Tang::AstNodeArray, [22](#)
 - Tang::AstNodeAssign, [26](#)
 - Tang::AstNodeBinary, [30](#)
 - Tang::AstNodeBlock, [35](#)
 - Tang::AstNodeBoolean, [39](#)
 - Tang::AstNodeBreak, [42](#)
 - Tang::AstNodeCast, [47](#)
 - Tang::AstNodeContinue, [51](#)
 - Tang::AstNodeDoWhile, [56](#)
 - Tang::AstNodeFloat, [60](#)
 - Tang::AstNodeFor, [64](#)
 - Tang::AstNodeFunctionCall, [68](#)
 - Tang::AstNodeFunctionDeclaration, [71](#)
 - Tang::AstNodeIdentifier, [76](#)
 - Tang::AstNodeIfElse, [81](#)
 - Tang::AstNodeIndex, [86](#)
 - Tang::AstNodeInteger, [91](#)
 - Tang::AstNodePrint, [95](#)
 - Tang::AstNodeRangedFor, [98](#)
 - Tang::AstNodeReturn, [103](#)
 - Tang::AstNodeSlice, [108](#)
 - Tang::AstNodeString, [112](#)
 - Tang::AstNodeTernary, [118](#)
 - Tang::AstNodeUnary, [122](#)
 - Tang::AstNodeWhile, [127](#)

- compileLiteral
 - Tang::AstNodeString, [113](#)
- compilePreprocess
 - Tang::AstNode, [17](#)
 - Tang::AstNodeArray, [22](#)
 - Tang::AstNodeAssign, [26](#)
 - Tang::AstNodeBinary, [31](#)
 - Tang::AstNodeBlock, [35](#)
 - Tang::AstNodeBoolean, [39](#)
 - Tang::AstNodeBreak, [43](#)
 - Tang::AstNodeCast, [47](#)
 - Tang::AstNodeContinue, [51](#)
 - Tang::AstNodeDoWhile, [56](#)
 - Tang::AstNodeFloat, [60](#)
 - Tang::AstNodeFor, [65](#)
 - Tang::AstNodeFunctionCall, [68](#)
 - Tang::AstNodeFunctionDeclaration, [72](#)
 - Tang::AstNodeIdentifier, [76](#)
 - Tang::AstNodeIfElse, [82](#)
 - Tang::AstNodeIndex, [86](#)
 - Tang::AstNodeInteger, [91](#)
 - Tang::AstNodePrint, [95](#)
 - Tang::AstNodeRangedFor, [99](#)
 - Tang::AstNodeReturn, [103](#)
 - Tang::AstNodeSlice, [108](#)
 - Tang::AstNodeString, [113](#)
 - Tang::AstNodeTernary, [118](#)
 - Tang::AstNodeUnary, [123](#)
 - Tang::AstNodeWhile, [127](#)
- compileScript
 - Tang::TangBase, [304](#)
- ComputedExpressionArray
 - Tang::ComputedExpressionArray, [144](#)
- ComputedExpressionBoolean
 - Tang::ComputedExpressionBoolean, [157](#)
- ComputedExpressionCompiledFunction
 - Tang::ComputedExpressionCompiledFunction, [169](#)
- ComputedExpressionError
 - Tang::ComputedExpressionError, [183](#)
- ComputedExpressionFloat
 - Tang::ComputedExpressionFloat, [195](#)
- ComputedExpressionInteger
 - Tang::ComputedExpressionInteger, [210](#)
- ComputedExpressionIterator
 - Tang::ComputedExpressionIterator, [225](#)
- ComputedExpressionString
 - Tang::ComputedExpressionString, [251](#)
- COPY
 - opcode.hpp, [358](#)
- currentIndex
 - Tang::SingletonObjectPool< T >, [303](#)
- currentRecycledIndex
 - Tang::SingletonObjectPool< T >, [303](#)
- Default
 - Tang::AstNode, [16](#)
 - Tang::AstNodeArray, [21](#)
 - Tang::AstNodeAssign, [25](#)
 - Tang::AstNodeBinary, [30](#)
 - Tang::AstNodeBlock, [34](#)
 - Tang::AstNodeBoolean, [38](#)
 - Tang::AstNodeBreak, [42](#)
 - Tang::AstNodeCast, [46](#)
 - Tang::AstNodeContinue, [50](#)
 - Tang::AstNodeDoWhile, [55](#)
 - Tang::AstNodeFloat, [59](#)
 - Tang::AstNodeFor, [63](#)
 - Tang::AstNodeFunctionCall, [67](#)
 - Tang::AstNodeFunctionDeclaration, [71](#)
 - Tang::AstNodeIdentifier, [75](#)
 - Tang::AstNodeIfElse, [80](#)
 - Tang::AstNodeIndex, [85](#)
 - Tang::AstNodeInteger, [90](#)
 - Tang::AstNodePrint, [94](#)
 - Tang::AstNodeRangedFor, [98](#)
 - Tang::AstNodeReturn, [102](#)
 - Tang::AstNodeSlice, [107](#)
 - Tang::AstNodeString, [111](#)
 - Tang::AstNodeTernary, [117](#)
 - Tang::AstNodeUnary, [122](#)
 - Tang::AstNodeWhile, [126](#)
- DIVIDE
 - opcode.hpp, [358](#)
- Divide
 - Tang::AstNodeBinary, [29](#)
- dump
 - Tang::AstNode, [18](#)
 - Tang::AstNodeArray, [23](#)
 - Tang::AstNodeAssign, [27](#)
 - Tang::AstNodeBinary, [31](#)
 - Tang::AstNodeBlock, [35](#)
 - Tang::AstNodeBoolean, [39](#)
 - Tang::AstNodeBreak, [43](#)
 - Tang::AstNodeCast, [48](#)
 - Tang::AstNodeContinue, [52](#)
 - Tang::AstNodeDoWhile, [57](#)
 - Tang::AstNodeFloat, [61](#)
 - Tang::AstNodeFor, [65](#)
 - Tang::AstNodeFunctionCall, [69](#)
 - Tang::AstNodeFunctionDeclaration, [72](#)
 - Tang::AstNodeIdentifier, [77](#)
 - Tang::AstNodeIfElse, [82](#)
 - Tang::AstNodeIndex, [87](#)
 - Tang::AstNodeInteger, [92](#)
 - Tang::AstNodePrint, [96](#)
 - Tang::AstNodeRangedFor, [100](#)
 - Tang::AstNodeReturn, [104](#)
 - Tang::AstNodeSlice, [109](#)
 - Tang::AstNodeString, [114](#)
 - Tang::AstNodeTernary, [119](#)
 - Tang::AstNodeUnary, [123](#)
 - Tang::AstNodeWhile, [128](#)
 - Tang::ComputedExpression, [137](#)
 - Tang::ComputedExpressionArray, [151](#)
 - Tang::ComputedExpressionBoolean, [163](#)
 - Tang::ComputedExpressionCompiledFunction, [176](#)
 - Tang::ComputedExpressionError, [189](#)

- Tang::ComputedExpressionFloat, [204](#)
- Tang::ComputedExpressionInteger, [220](#)
- Tang::ComputedExpressionIterator, [233](#)
- Tang::ComputedExpressionIteratorEnd, [245](#)
- Tang::ComputedExpressionString, [260](#)
- dumpBytecode
 - Tang::Program, [296](#)
- DUMPPROGRAMCHECK
 - program-dumpBytecode.cpp, [395](#)
- EQ
 - opcode.hpp, [358](#)
- Equal
 - Tang::AstNodeBinary, [29](#)
- Error
 - Tang::Error, [265](#)
- error.cpp
 - operator<<, [394](#)
- execute
 - Tang::Program, [296](#)
- EXECUTEPROGRAMCHECK
 - program-execute.cpp, [397](#)
- FLOAT
 - opcode.hpp, [358](#)
- Float
 - Tang::AstNodeCast, [46](#)
- FUNCTION
 - opcode.hpp, [358](#)
- functionsDeclared
 - Tang::Program, [300](#)
- GarbageCollected
 - Tang::GarbageCollected, [269](#)
- get
 - Tang::SingletonObjectPool< T >, [302](#)
- get_next_token
 - Tang::HtmlEscape, [285](#)
 - Tang::HtmlEscapeAscii, [287](#)
 - Tang::TangScanner, [307](#)
 - Tang::Unescape, [309](#)
- getAst
 - Tang::Program, [296](#)
- getBytecode
 - Tang::Program, [296](#)
- getCode
 - Tang::Program, [297](#)
- getCollection
 - Tang::AstNodeIndex, [87](#)
- getIdentifiers
 - Tang::Program, [297](#)
- getIdentifiersAssigned
 - Tang::Program, [297](#)
- getIndex
 - Tang::AstNodeIndex, [87](#)
- getInstance
 - Tang::SingletonObjectPool< T >, [302](#)
- GETITERATOR
 - opcode.hpp, [358](#)
- getResult
 - Tang::Program, [297](#)
- getStrings
 - Tang::Program, [298](#)
- getValue
 - Tang::ComputedExpressionFloat, [205](#)
 - Tang::ComputedExpressionInteger, [220](#)
- GreaterThan
 - Tang::AstNodeBinary, [29](#)
- GreaterThanEqual
 - Tang::AstNodeBinary, [29](#)
- GT
 - opcode.hpp, [358](#)
- GTE
 - opcode.hpp, [358](#)
- HtmlEscape
 - Tang::HtmlEscape, [284](#)
- htmlEscape
 - unicodeString.hpp, [366](#)
- HtmlEscapeAscii
 - Tang::HtmlEscapeAscii, [286](#)
- htmlEscapeAscii
 - unicodeString.hpp, [366](#)
- include/astNode.hpp, [317](#)
- include/astNodeArray.hpp, [318](#)
- include/astNodeAssign.hpp, [319](#)
- include/astNodeBinary.hpp, [320](#)
- include/astNodeBlock.hpp, [321](#)
- include/astNodeBoolean.hpp, [322](#)
- include/astNodeBreak.hpp, [323](#)
- include/astNodeCast.hpp, [324](#)
- include/astNodeContinue.hpp, [325](#)
- include/astNodeDoWhile.hpp, [326](#)
- include/astNodeFloat.hpp, [327](#)
- include/astNodeFor.hpp, [328](#)
- include/astNodeFunctionCall.hpp, [329](#)
- include/astNodeFunctionDeclaration.hpp, [330](#)
- include/astNodeIdentifier.hpp, [331](#)
- include/astNodeIfElse.hpp, [332](#)
- include/astNodeIndex.hpp, [333](#)
- include/astNodeInteger.hpp, [334](#)
- include/astNodePrint.hpp, [335](#)
- include/astNodeRangedFor.hpp, [336](#)
- include/astNodeReturn.hpp, [337](#)
- include/astNodeSlice.hpp, [338](#)
- include/astNodeString.hpp, [339](#)
- include/astNodeTernary.hpp, [340](#)
- include/astNodeUnary.hpp, [341](#)
- include/astNodeWhile.hpp, [342](#)
- include/computedExpression.hpp, [343](#)
- include/computedExpressionArray.hpp, [344](#)
- include/computedExpressionBoolean.hpp, [345](#)
- include/computedExpressionCompiledFunction.hpp, [346](#)
- include/computedExpressionError.hpp, [347](#)
- include/computedExpressionFloat.hpp, [348](#)
- include/computedExpressionInteger.hpp, [349](#)

- include/computedExpressionIterator.hpp, 350
- include/computedExpressionIteratorEnd.hpp, 351
- include/computedExpressionString.hpp, 352
- include/error.hpp, 352
- include/garbageCollected.hpp, 353
- include/htmlEscape.hpp, 354
- include/htmlEscapeAscii.hpp, 356
- include/macros.hpp, 357
- include/opcode.hpp, 357
- include/program.hpp, 359
- include/singletonObjectPool.hpp, 360
- include/tang.hpp, 361
- include/tangBase.hpp, 362
- include/tangScanner.hpp, 363
- include/unescape.hpp, 364
- include/unicodeString.hpp, 365
- INDEX
 - opcode.hpp, 358
- INTEGER
 - opcode.hpp, 358
- Integer
 - Tang::AstNodeCast, 46
- is_equal
 - Tang::ComputedExpression, 138–140
 - Tang::ComputedExpressionArray, 151–153
 - Tang::ComputedExpressionBoolean, 164–166
 - Tang::ComputedExpressionCompiledFunction, 176–178
 - Tang::ComputedExpressionError, 190–192
 - Tang::ComputedExpressionFloat, 205–207
 - Tang::ComputedExpressionInteger, 220–222
 - Tang::ComputedExpressionIterator, 233, 235, 236
 - Tang::ComputedExpressionIteratorEnd, 246–248
 - Tang::ComputedExpressionString, 261–263
- IsAssignment
 - Tang::AstNode, 16
 - Tang::AstNodeArray, 21
 - Tang::AstNodeAssign, 25
 - Tang::AstNodeBinary, 30
 - Tang::AstNodeBlock, 34
 - Tang::AstNodeBoolean, 38
 - Tang::AstNodeBreak, 42
 - Tang::AstNodeCast, 46
 - Tang::AstNodeContinue, 50
 - Tang::AstNodeDoWhile, 55
 - Tang::AstNodeFloat, 59
 - Tang::AstNodeFor, 63
 - Tang::AstNodeFunctionCall, 67
 - Tang::AstNodeFunctionDeclaration, 71
 - Tang::AstNodeIdentifier, 75
 - Tang::AstNodeIfElse, 80
 - Tang::AstNodeIndex, 85
 - Tang::AstNodeInteger, 90
 - Tang::AstNodePrint, 94
 - Tang::AstNodeRangedFor, 98
 - Tang::AstNodeReturn, 102
 - Tang::AstNodeSlice, 107
 - Tang::AstNodeString, 111
 - Tang::AstNodeTernary, 117
 - Tang::AstNodeUnary, 122
 - Tang::AstNodeWhile, 126
- isCopyNeeded
 - Tang::ComputedExpression, 140
 - Tang::ComputedExpressionArray, 154
 - Tang::ComputedExpressionBoolean, 166
 - Tang::ComputedExpressionCompiledFunction, 179
 - Tang::ComputedExpressionError, 192
 - Tang::ComputedExpressionFloat, 207
 - Tang::ComputedExpressionInteger, 222
 - Tang::ComputedExpressionIterator, 236
 - Tang::ComputedExpressionIteratorEnd, 248
 - Tang::ComputedExpressionString, 263
 - Tang::GarbageCollected, 270
- ISITERATOREND
 - opcode.hpp, 359
- ITERATORNEXT
 - opcode.hpp, 359
- JMP
 - opcode.hpp, 358
- JMPF
 - opcode.hpp, 358
- JMPF_POP
 - opcode.hpp, 358
- JMPT
 - opcode.hpp, 358
- JMPT_POP
 - opcode.hpp, 358
- length
 - Tang::UnicodeString, 311
- LessThan
 - Tang::AstNodeBinary, 29
- LessThanEqual
 - Tang::AstNodeBinary, 29
- location.hh
 - operator<<, 316, 317
- LT
 - opcode.hpp, 358
- LTE
 - opcode.hpp, 358
- make
 - Tang::GarbageCollected, 270
- makeCopy
 - Tang::ComputedExpression, 140
 - Tang::ComputedExpressionArray, 154
 - Tang::ComputedExpressionBoolean, 166
 - Tang::ComputedExpressionCompiledFunction, 179
 - Tang::ComputedExpressionError, 192
 - Tang::ComputedExpressionFloat, 207
 - Tang::ComputedExpressionInteger, 223
 - Tang::ComputedExpressionIterator, 236
 - Tang::ComputedExpressionIteratorEnd, 248
 - Tang::ComputedExpressionString, 263
 - Tang::GarbageCollected, 271
- MODULO

- opcode.hpp, 358
- Modulo
 - Tang::AstNodeBinary, 29
- MULTIPLY
 - opcode.hpp, 358
- Multiply
 - Tang::AstNodeBinary, 29
- NEGATIVE
 - opcode.hpp, 358
- Negative
 - Tang::AstNodeUnary, 121
- NEQ
 - opcode.hpp, 358
- NOT
 - opcode.hpp, 358
- Not
 - Tang::AstNodeUnary, 121
- NotEqual
 - Tang::AstNodeBinary, 29
- NULLVAL
 - opcode.hpp, 358
- Opcode
 - opcode.hpp, 358
- opcode.hpp
 - ADD, 358
 - ARRAY, 358
 - ASSIGNINDEX, 358
 - BOOLEAN, 358
 - CALLFUNC, 359
 - CASTBOOLEAN, 359
 - CASTFLOAT, 359
 - CASTINTEGER, 359
 - COPY, 358
 - DIVIDE, 358
 - EQ, 358
 - FLOAT, 358
 - FUNCTION, 358
 - GETITERATOR, 358
 - GT, 358
 - GTE, 358
 - INDEX, 358
 - INTEGER, 358
 - ISITERATOREND, 359
 - ITERATORNEXT, 359
 - JMP, 358
 - JMPF, 358
 - JMPF_POP, 358
 - JMPT, 358
 - JMPT_POP, 358
 - LT, 358
 - LTE, 358
 - MODULO, 358
 - MULTIPLY, 358
 - NEGATIVE, 358
 - NEQ, 358
 - NOT, 358
 - NULLVAL, 358
 - Opcode, 358
 - PEEK, 358
 - POKE, 358
 - POP, 358
 - PRINT, 359
 - RETURN, 359
 - SLICE, 358
 - STRING, 358
 - SUBTRACT, 358
- Operation
 - Tang::AstNodeBinary, 29
- Operator
 - Tang::AstNodeUnary, 121
- operator std::string
 - Tang::UnicodeString, 312
- operator!
 - Tang::GarbageCollected, 271
- operator!=
 - Tang::GarbageCollected, 272
- operator<
 - Tang::GarbageCollected, 276
 - Tang::UnicodeString, 313
- operator<<
 - error.cpp, 394
 - location.hh, 316, 317
 - Tang::Error, 266
 - Tang::GarbageCollected, 283
- operator<=
 - Tang::GarbageCollected, 277
- operator>
 - Tang::GarbageCollected, 282
- operator>=
 - Tang::GarbageCollected, 282
- operator*
 - Tang::GarbageCollected, 273
- operator+
 - Tang::GarbageCollected, 274
 - Tang::UnicodeString, 312
- operator-
 - Tang::GarbageCollected, 274, 275
- operator->
 - Tang::GarbageCollected, 275
- operator/
 - Tang::GarbageCollected, 276
- operator=
 - Tang::GarbageCollected, 277
- operator==
 - Tang::GarbageCollected, 279–281
 - Tang::UnicodeString, 313
- operator%
 - Tang::GarbageCollected, 272
- Or
 - Tang::AstNodeBinary, 29
- PEEK
 - opcode.hpp, 358
- POKE
 - opcode.hpp, 358
- POP

- opcode.hpp, 358
- popBreakStack
 - Tang::Program, 298
- popContinueStack
 - Tang::Program, 298
- PreprocessState
 - Tang::AstNode, 16
 - Tang::AstNodeArray, 21
 - Tang::AstNodeAssign, 25
 - Tang::AstNodeBinary, 30
 - Tang::AstNodeBlock, 34
 - Tang::AstNodeBoolean, 38
 - Tang::AstNodeBreak, 42
 - Tang::AstNodeCast, 46
 - Tang::AstNodeContinue, 50
 - Tang::AstNodeDoWhile, 55
 - Tang::AstNodeFloat, 59
 - Tang::AstNodeFor, 63
 - Tang::AstNodeFunctionCall, 67
 - Tang::AstNodeFunctionDeclaration, 70
 - Tang::AstNodeIdentifier, 75
 - Tang::AstNodeIfElse, 80
 - Tang::AstNodeIndex, 85
 - Tang::AstNodeInteger, 90
 - Tang::AstNodePrint, 94
 - Tang::AstNodeRangedFor, 98
 - Tang::AstNodeReturn, 102
 - Tang::AstNodeSlice, 107
 - Tang::AstNodeString, 111
 - Tang::AstNodeTernary, 117
 - Tang::AstNodeUnary, 122
 - Tang::AstNodeWhile, 126
- PRINT
 - opcode.hpp, 359
- Program
 - Tang::Program, 293
- program-dumpBytecode.cpp
 - DUMPPROGRAMCHECK, 395
- program-execute.cpp
 - EXECUTEPROGRAMCHECK, 397
 - STACKCHECK, 397
- pushEnvironment
 - Tang::Program, 299
- recycle
 - Tang::SingletonObjectPool< T >, 303
- RETURN
 - opcode.hpp, 359
- Script
 - Tang::Program, 293
- setFunctionStackDeclaration
 - Tang::Program, 299
- setJumpTarget
 - Tang::Program, 300
- SLICE
 - opcode.hpp, 358
- src/astNode.cpp, 368
- src/astNodeArray.cpp, 368
- src/astNodeAssign.cpp, 369
- src/astNodeBinary.cpp, 370
- src/astNodeBlock.cpp, 371
- src/astNodeBoolean.cpp, 371
- src/astNodeBreak.cpp, 372
- src/astNodeCast.cpp, 373
- src/astNodeContinue.cpp, 373
- src/astNodeDoWhile.cpp, 374
- src/astNodeFloat.cpp, 375
- src/astNodeFor.cpp, 376
- src/astNodeFunctionCall.cpp, 376
- src/astNodeFunctionDeclaration.cpp, 377
- src/astNodeIdentifier.cpp, 378
- src/astNodeIfElse.cpp, 379
- src/astNodeIndex.cpp, 379
- src/astNodeInteger.cpp, 380
- src/astNodePrint.cpp, 381
- src/astNodeRangedFor.cpp, 381
- src/astNodeReturn.cpp, 382
- src/astNodeSlice.cpp, 383
- src/astNodeString.cpp, 384
- src/astNodeTernary.cpp, 385
- src/astNodeUnary.cpp, 386
- src/astNodeWhile.cpp, 386
- src/computedExpression.cpp, 387
- src/computedExpressionArray.cpp, 388
- src/computedExpressionBoolean.cpp, 389
- src/computedExpressionCompiledFunction.cpp, 389
- src/computedExpressionError.cpp, 390
- src/computedExpressionFloat.cpp, 391
- src/computedExpressionInteger.cpp, 391
- src/computedExpressionIterator.cpp, 392
- src/computedExpressionIteratorEnd.cpp, 393
- src/computedExpressionString.cpp, 393
- src/error.cpp, 394
- src/program-dumpBytecode.cpp, 395
- src/program-execute.cpp, 396
- src/program.cpp, 397
- src/tangBase.cpp, 398
- src/unicodeString.cpp, 399
- STACKCHECK
 - program-execute.cpp, 397
- STRING
 - opcode.hpp, 358
- substr
 - Tang::UnicodeString, 313
- SUBTRACT
 - opcode.hpp, 358
- Subtract
 - Tang::AstNodeBinary, 29
- Tang::AstNode, 13
 - AstNode, 16
 - compile, 17
 - compilePreprocess, 17
 - Default, 16
 - dump, 18
 - IsAssignment, 16
 - PreprocessState, 16

- Tang::AstNodeArray, 18
 - AstNodeArray, 21
 - compile, 22
 - compilePreprocess, 22
 - Default, 21
 - dump, 23
 - IsAssignment, 21
 - PreprocessState, 21
- Tang::AstNodeAssign, 23
 - AstNodeAssign, 25
 - compile, 26
 - compilePreprocess, 26
 - Default, 25
 - dump, 27
 - IsAssignment, 25
 - PreprocessState, 25
- Tang::AstNodeBinary, 27
 - Add, 29
 - And, 29
 - AstNodeBinary, 30
 - compile, 30
 - compilePreprocess, 31
 - Default, 30
 - Divide, 29
 - dump, 31
 - Equal, 29
 - GreaterThan, 29
 - GreaterThanEqual, 29
 - IsAssignment, 30
 - LessThan, 29
 - LessThanEqual, 29
 - Modulo, 29
 - Multiply, 29
 - NotEqual, 29
 - Operation, 29
 - Or, 29
 - PreprocessState, 30
 - Subtract, 29
- Tang::AstNodeBlock, 32
 - AstNodeBlock, 34
 - compile, 35
 - compilePreprocess, 35
 - Default, 34
 - dump, 35
 - IsAssignment, 34
 - PreprocessState, 34
- Tang::AstNodeBoolean, 36
 - AstNodeBoolean, 38
 - compile, 39
 - compilePreprocess, 39
 - Default, 38
 - dump, 39
 - IsAssignment, 38
 - PreprocessState, 38
- Tang::AstNodeBreak, 40
 - AstNodeBreak, 42
 - compile, 42
 - compilePreprocess, 43
 - Default, 42
 - dump, 43
 - IsAssignment, 42
 - PreprocessState, 42
- Tang::AstNodeCast, 44
 - AstNodeCast, 47
 - Boolean, 46
 - compile, 47
 - compilePreprocess, 47
 - Default, 46
 - dump, 48
 - Float, 46
 - Integer, 46
 - IsAssignment, 46
 - PreprocessState, 46
 - Type, 46
- Tang::AstNodeContinue, 48
 - AstNodeContinue, 50
 - compile, 51
 - compilePreprocess, 51
 - Default, 50
 - dump, 52
 - IsAssignment, 50
 - PreprocessState, 50
- Tang::AstNodeDoWhile, 52
 - AstNodeDoWhile, 55
 - compile, 56
 - compilePreprocess, 56
 - Default, 55
 - dump, 57
 - IsAssignment, 55
 - PreprocessState, 55
- Tang::AstNodeFloat, 57
 - AstNodeFloat, 59
 - compile, 60
 - compilePreprocess, 60
 - Default, 59
 - dump, 61
 - IsAssignment, 59
 - PreprocessState, 59
- Tang::AstNodeFor, 61
 - AstNodeFor, 63
 - compile, 64
 - compilePreprocess, 65
 - Default, 63
 - dump, 65
 - IsAssignment, 63
 - PreprocessState, 63
- Tang::AstNodeFunctionCall, 66
 - AstNodeFunctionCall, 67
 - compile, 68
 - compilePreprocess, 68
 - Default, 67
 - dump, 69
 - IsAssignment, 67
 - PreprocessState, 67
- Tang::AstNodeFunctionDeclaration, 69
 - AstNodeFunctionDeclaration, 71

- compile, 71
- compilePreprocess, 72
- Default, 71
- dump, 72
- IsAssignment, 71
- PreprocessState, 70
- Tang::AstNodeIdentifier, 73
 - AstNodeIdentifier, 75
 - compile, 76
 - compilePreprocess, 76
 - Default, 75
 - dump, 77
 - IsAssignment, 75
 - PreprocessState, 75
- Tang::AstNodeIfElse, 77
 - AstNodeIfElse, 80, 81
 - compile, 81
 - compilePreprocess, 82
 - Default, 80
 - dump, 82
 - IsAssignment, 80
 - PreprocessState, 80
- Tang::AstNodeIndex, 83
 - AstNodeIndex, 85
 - compile, 86
 - compilePreprocess, 86
 - Default, 85
 - dump, 87
 - getCollection, 87
 - getIndex, 87
 - IsAssignment, 85
 - PreprocessState, 85
- Tang::AstNodeInteger, 88
 - AstNodeInteger, 90
 - compile, 91
 - compilePreprocess, 91
 - Default, 90
 - dump, 92
 - IsAssignment, 90
 - PreprocessState, 90
- Tang::AstNodePrint, 92
 - AstNodePrint, 95
 - compile, 95
 - compilePreprocess, 95
 - Default, 94
 - dump, 96
 - IsAssignment, 94
 - PreprocessState, 94
 - Type, 94
- Tang::AstNodeRangedFor, 96
 - AstNodeRangedFor, 98
 - compile, 98
 - compilePreprocess, 99
 - Default, 98
 - dump, 100
 - IsAssignment, 98
 - PreprocessState, 98
- Tang::AstNodeReturn, 100
 - AstNodeReturn, 102
 - compile, 103
 - compilePreprocess, 103
 - Default, 102
 - dump, 104
 - IsAssignment, 102
 - PreprocessState, 102
- Tang::AstNodeSlice, 104
 - AstNodeSlice, 107
 - compile, 108
 - compilePreprocess, 108
 - Default, 107
 - dump, 109
 - IsAssignment, 107
 - PreprocessState, 107
- Tang::AstNodeString, 109
 - AstNodeString, 112
 - compile, 112
 - compileLiteral, 113
 - compilePreprocess, 113
 - Default, 111
 - dump, 114
 - IsAssignment, 111
 - PreprocessState, 111
- Tang::AstNodeTernary, 114
 - AstNodeTernary, 117
 - compile, 118
 - compilePreprocess, 118
 - Default, 117
 - dump, 119
 - IsAssignment, 117
 - PreprocessState, 117
- Tang::AstNodeUnary, 119
 - AstNodeUnary, 122
 - compile, 122
 - compilePreprocess, 123
 - Default, 122
 - dump, 123
 - IsAssignment, 122
 - Negative, 121
 - Not, 121
 - Operator, 121
 - PreprocessState, 122
- Tang::AstNodeWhile, 124
 - AstNodeWhile, 126
 - compile, 127
 - compilePreprocess, 127
 - Default, 126
 - dump, 128
 - IsAssignment, 126
 - PreprocessState, 126
- Tang::ComputedExpression, 128
 - __add, 131
 - __asCode, 131
 - __assign_index, 131
 - __boolean, 132
 - __divide, 132
 - __equal, 132

- __float, 133
- __getIterator, 133
- __index, 133
- __integer, 134
- __iteratorNext, 134
- __lessThan, 134
- __modulo, 135
- __multiply, 135
- __negative, 135
- __not, 136
- __slice, 136
- __string, 137
- __subtract, 137
- dump, 137
- is_equal, 138–140
- isCopyNeeded, 140
- makeCopy, 140
- Tang::ComputedExpressionArray, 141
 - __add, 144
 - __asCode, 144
 - __assign_index, 144
 - __boolean, 145
 - __divide, 145
 - __equal, 146
 - __float, 146
 - __getIterator, 146
 - __index, 147
 - __integer, 147
 - __iteratorNext, 147
 - __lessThan, 148
 - __modulo, 148
 - __multiply, 149
 - __negative, 149
 - __not, 149
 - __slice, 149
 - __string, 150
 - __subtract, 151
- ComputedExpressionArray, 144
- dump, 151
- is_equal, 151–153
- isCopyNeeded, 154
- makeCopy, 154
- Tang::ComputedExpressionBoolean, 155
 - __add, 157
 - __asCode, 157
 - __assign_index, 158
 - __boolean, 158
 - __divide, 158
 - __equal, 159
 - __float, 159
 - __getIterator, 159
 - __index, 160
 - __integer, 160
 - __iteratorNext, 160
 - __lessThan, 161
 - __modulo, 161
 - __multiply, 161
 - __negative, 162
 - __not, 162
 - __slice, 162
 - __string, 163
 - __subtract, 163
- ComputedExpressionBoolean, 157
- dump, 163
- is_equal, 164–166
- isCopyNeeded, 166
- makeCopy, 166
- Tang::ComputedExpressionCompiledFunction, 167
 - __add, 169
 - __asCode, 170
 - __assign_index, 170
 - __boolean, 170
 - __divide, 171
 - __equal, 171
 - __float, 172
 - __getIterator, 172
 - __index, 172
 - __integer, 173
 - __iteratorNext, 173
 - __lessThan, 173
 - __modulo, 174
 - __multiply, 174
 - __negative, 174
 - __not, 175
 - __slice, 175
 - __string, 176
 - __subtract, 176
- ComputedExpressionCompiledFunction, 169
- dump, 176
- is_equal, 176–178
- isCopyNeeded, 179
- makeCopy, 179
- Tang::ComputedExpressionError, 180
 - __add, 183
 - __asCode, 183
 - __assign_index, 184
 - __boolean, 184
 - __divide, 184
 - __equal, 185
 - __float, 185
 - __getIterator, 185
 - __index, 186
 - __integer, 186
 - __iteratorNext, 186
 - __lessThan, 187
 - __modulo, 187
 - __multiply, 187
 - __negative, 188
 - __not, 188
 - __slice, 188
 - __string, 189
 - __subtract, 189
- ComputedExpressionError, 183
- dump, 189
- is_equal, 190–192
- isCopyNeeded, 192

- makeCopy, 192
- Tang::ComputedExpressionFloat, 193
 - __add, 195
 - __asCode, 197
 - __assign_index, 197
 - __boolean, 198
 - __divide, 198
 - __equal, 198
 - __float, 199
 - __getIterator, 199
 - __index, 200
 - __integer, 200
 - __iteratorNext, 200
 - __lessThan, 201
 - __modulo, 201
 - __multiply, 202
 - __negative, 202
 - __not, 202
 - __slice, 203
 - __string, 203
 - __subtract, 204
- ComputedExpressionFloat, 195
- dump, 204
- getValue, 205
- is_equal, 205–207
- isCopyNeeded, 207
- makeCopy, 207
- Tang::ComputedExpressionInteger, 208
 - __add, 210
 - __asCode, 212
 - __assign_index, 212
 - __boolean, 213
 - __divide, 213
 - __equal, 213
 - __float, 214
 - __getIterator, 214
 - __index, 215
 - __integer, 215
 - __iteratorNext, 215
 - __lessThan, 216
 - __modulo, 216
 - __multiply, 217
 - __negative, 217
 - __not, 218
 - __slice, 218
 - __string, 219
 - __subtract, 219
- ComputedExpressionInteger, 210
- dump, 220
- getValue, 220
- is_equal, 220–222
- isCopyNeeded, 222
- makeCopy, 223
- Tang::ComputedExpressionIterator, 223
 - __add, 226
 - __asCode, 226
 - __assign_index, 226
 - __boolean, 227
 - __divide, 227
 - __equal, 228
 - __float, 228
 - __getIterator, 228
 - __index, 229
 - __integer, 229
 - __iteratorNext, 229
 - __lessThan, 230
 - __modulo, 230
 - __multiply, 231
 - __negative, 231
 - __not, 231
 - __slice, 231
 - __string, 232
 - __subtract, 232
- ComputedExpressionIterator, 225
- dump, 233
- is_equal, 233, 235, 236
- isCopyNeeded, 236
- makeCopy, 236
- Tang::ComputedExpressionIteratorEnd, 237
 - __add, 239
 - __asCode, 239
 - __assign_index, 239
 - __boolean, 240
 - __divide, 240
 - __equal, 241
 - __float, 241
 - __getIterator, 241
 - __index, 242
 - __integer, 242
 - __iteratorNext, 242
 - __lessThan, 243
 - __modulo, 243
 - __multiply, 243
 - __negative, 244
 - __not, 244
 - __slice, 244
 - __string, 245
 - __subtract, 245
- dump, 245
- is_equal, 246–248
- isCopyNeeded, 248
- makeCopy, 248
- Tang::ComputedExpressionString, 249
 - __add, 252
 - __asCode, 252
 - __assign_index, 253
 - __boolean, 253
 - __divide, 254
 - __equal, 254
 - __float, 255
 - __getIterator, 255
 - __index, 255
 - __integer, 256
 - __iteratorNext, 256
 - __lessThan, 257
 - __modulo, 257

- __multiply, [258](#)
- __negative, [258](#)
- __not, [258](#)
- __slice, [259](#)
- __string, [260](#)
- __subtract, [260](#)
- ComputedExpressionString, [251](#)
- dump, [260](#)
- is_equal, [261–263](#)
- isCopyNeeded, [263](#)
- makeCopy, [263](#)
- Tang::Error, [264](#)
 - Error, [265](#)
 - operator<<, [266](#)
- Tang::GarbageCollected, [266](#)
 - ~GarbageCollected, [269](#)
 - GarbageCollected, [269](#)
 - isCopyNeeded, [270](#)
 - make, [270](#)
 - makeCopy, [271](#)
 - operator!, [271](#)
 - operator!=, [272](#)
 - operator<, [276](#)
 - operator<<, [283](#)
 - operator<=, [277](#)
 - operator>, [282](#)
 - operator>=, [282](#)
 - operator*, [273](#)
 - operator+, [274](#)
 - operator-, [274, 275](#)
 - operator->, [275](#)
 - operator/, [276](#)
 - operator=, [277](#)
 - operator==, [279–281](#)
 - operator%, [272](#)
- Tang::HtmlEscape, [283](#)
 - get_next_token, [285](#)
 - HtmlEscape, [284](#)
- Tang::HtmlEscapeAscii, [285](#)
 - get_next_token, [287](#)
 - HtmlEscapeAscii, [286](#)
- Tang::location, [287](#)
- Tang::position, [289](#)
- Tang::Program, [290](#)
 - addBreak, [294](#)
 - addBytecode, [294](#)
 - addContinue, [294](#)
 - addIdentifier, [295](#)
 - addIdentifierAssigned, [295](#)
 - addString, [295](#)
 - CodeType, [293](#)
 - dumpBytecode, [296](#)
 - execute, [296](#)
 - functionsDeclared, [300](#)
 - getAst, [296](#)
 - getBytecode, [296](#)
 - getCode, [297](#)
 - getIdentifiers, [297](#)
 - getIdentifiersAssigned, [297](#)
 - getResult, [297](#)
 - getStrings, [298](#)
 - popBreakStack, [298](#)
 - popContinueStack, [298](#)
 - Program, [293](#)
 - pushEnvironment, [299](#)
 - Script, [293](#)
 - setFunctionStackDeclaration, [299](#)
 - setJumpTarget, [300](#)
 - Template, [293](#)
- Tang::SingletonObjectPool< T >, [301](#)
 - currentIndex, [303](#)
 - currentRecycledIndex, [303](#)
 - get, [302](#)
 - getInstance, [302](#)
 - recycle, [303](#)
- Tang::TangBase, [304](#)
 - compileScript, [304](#)
 - TangBase, [304](#)
- Tang::TangScanner, [305](#)
 - get_next_token, [307](#)
 - TangScanner, [307](#)
- Tang::Unescape, [308](#)
 - get_next_token, [309](#)
 - Unescape, [309](#)
- Tang::UnicodeString, [310](#)
 - bytesLength, [311](#)
 - length, [311](#)
 - operator std::string, [312](#)
 - operator<, [313](#)
 - operator+, [312](#)
 - operator==, [313](#)
 - substr, [313](#)
 - UnicodeString, [311](#)
- TangBase
 - Tang::TangBase, [304](#)
- TangScanner
 - Tang::TangScanner, [307](#)
- Template
 - Tang::Program, [293](#)
- test/test.cpp, [399](#)
- test/testGarbageCollected.cpp, [401](#)
- test/testSingletonObjectPool.cpp, [402](#)
- test/testUnicodeString.cpp, [403](#)
- Type
 - Tang::AstNodeCast, [46](#)
 - Tang::AstNodePrint, [94](#)
- Unescape
 - Tang::Unescape, [309](#)
- unescape
 - unicodeString.hpp, [367](#)
- UnicodeString
 - Tang::UnicodeString, [311](#)
- unicodeString.hpp
 - htmlEscape, [366](#)
 - htmlEscapeAscii, [366](#)
 - unescape, [367](#)